2010 年度 博士論文

粒子ベースボリュームレンダリングによる大規模不規則格子ボリュームデータ向け可視化手法に関する研究

京都大学 大学院
工学研究科 電気工学専攻 博士後期課程
河村 拓馬

概要

有限要素法のような任意の形状を解析領域とする計算からは、不規則な格子構造を持つ不規則格子ボリュームデータが得られる。そして昨今の計算機性能や数値計算技術の発展により、大規模な不規則格子を用いた数値計算が行われるようになった。これは、対象となる問題の高度化により高精度な数値計算結果が求められ、格子の空間分割の密度が高まったことが原因である。更に、分割された解析領域に対する計算手法が発展し、スーパーコンピュータや PC クラスタ等の分散環境下で計算された、分割された大規模不規則格子から構成されるボリュームデータが出力されるようになった。

ボリュームレンダリングによる可視化は医療や工学の分野で有効性が示されているが、大規模な不規則格子に対するボリュームレンダリング手法は未だに十分な技術が開発されておらず、研究が続けられている。ボリュームレンダリングはスカラー場を半透明の物体として描画する可視化手法であり、ユーザーがスカラー値に対する色や不透明度を割り当てることで興味のある値を強調することができる。不規則格子に対して画像を生成するためには、アルファブレンディングに起因する格子の探索やソート処理が必要とされる。この計算を行うためには格子の隣接情報が必要になり、大規模データの場合メモリを圧迫するという問題がある。それに加えて、領域分割されたボリュームデータの場合、データ間のアクセスに関わる問題が発生し、従来手法によるボリュームレンダリングは更に困難なものとなった。

粒子ベースボリュームレンダリング（Particle-based volume rendering, PBVR）とは、ボリュームレンダリングを行う上でボトルネックになっていた格子の探索やソーティングを必要としない手法であり、大規模な不規則格子ボリュームデータに柔軟に対応できる可能性がある。しかし、ユーザーが指定する不透明度に対して生成する画像を一意に決定できないという問題点が指摘されており、また、不規則格子に適用する手法も開発されていなかった。

本論文では粒子モデルを PBVR に導入することで画像生成に関する問題を解決し、大規模な不規則格子ボリュームデータに適用するための粒子生成手法を提案した。また、PBVR は高画質化を行うためにピクセルの分割処理（サブピクセ

ル処理）を行っており、そのためフレームバッファが巨大になるという問題点があった。この問題を解決するため、従来の高画質化手法と同等の画質が得られる画像重畳を用いた手法を提案した。そして粒子モデルを導入した PBVR を有限要素法で計算された実データに適用し、領域分割された大規模不規則格子ボリュームデータ（7200 万の六面体要素や、2600 万の四面体二次要素から構成されたボリュームデータ）をボリュームレンダリングすることに成功した。

従来手法によるボリュームレンダリングはサンプリング点や積分区間の離散化を行っているため、急峻に変化する不透明度が与えられた場合に画像にアーティファクトが生じるという問題点があった。粒子モデルを導入した PBVR も同様に、急峻に変化する不透明度に対して発生粒子数の計算と粒子生成が困難になり、画像にアーティファクトが生じた。この問題を解決するため、四面体格子に適用可能な高品位な粒子生成手法である、四面体向けのレイヤードサンプリング法を提案した。そしてその技術を発展させ、六面体格子に適用可能なレイヤードサンプリング法を提案した。これらの手法を実データに適用した結果、従来手法よりも高精度に生成粒子数を計算することが可能になり、アーティファクトを減少させることに成功した。

大規模不規則格子ボリュームデータに対する可視化技術の研究は今なお重要なテーマである。更に今後は大規模化のみならず複雑化も進み、それに対応した新たな可視化手法が要請されると考えられる。大規模不規則格子ボリュームデータを効率よく扱うことができる本手法は、今後の可視化手法を開発する上での基盤となることが期待される。

目次

第1章　序論

1.1　研究背景

人間は多量の数値データを、何の加工も工夫もなく一度に把握することは困難である。例えば、ある実験で得られた数値データが非常に少数であれば、物理単位や実験状況等を考慮に入れることで、それらを理解することは容易である。しかしデータの数が増加するにつれてデータ間の関連性や全体像の理解は難しくなり、数字から直接理解することは叶わなくなる。こうした問題に直面したときの対処法の一つは、多数のデータを統計処理によって、例えば平均や分散のような、データ全体の何らかの特徴を現す統計量を計算することである。これは多量の数値を人間が理解できるよう、少ない数値に減少させたとも言える。もう一つの対処法は、グラフや散布図といった手法を用いることである。データ群を描画することで数字一つ一つからは理解できなかったデータの全容を直感的に把握できるようになる。このような行為は広い意味でデータの可視化であり、目で見ることで現象の定性的な理解を得る事ができるようになる。

計算機のディスプレイ上に画像を生成するコンピュータグラフィックスの技術が開発されると、以前は紙の上で行っていた可視化がディスプレイ上で行われるようになった。計算機可視化技術（Computational Visualization）と呼ばれるこの技術は可視化を行うための一般的なツールとなり、1980 年代以降急速に発展した。ディスプレイ中に描画される空間も二次元から三次元に移行し、現在では実験やシミュレーションで得られた三次元のデータを、そのまま三次元で可視化することが一般的になっている。

米国科学財団 による ViSC（Visualization in Scientific Computing）レポート [1] が 1987 年に発行されてから情報可視化の重要性が広く認知されるようになった。コンピュータグラフィックス研究の国際会議 SIGGRAPH において多くの可視化技術が提案され、医用画像処理や数値シミュレーションの結果の可視化に利用されるようになった。

実験計測や数値シミュレーションの結果として得られたボリュームデータを可視化するために、断面表示や等値面表示など様々な技術が提案されたが、情報量が多く有用な可視化手法としてボリュームレンダリングが挙げられる。この技術はボリュームデータを雲や煙のように全体を透かして見せることで、ボリュームデータの内部の値の分布と、その形状を一度に把握できる可視化手法である。しかしこの技術は計算量が多く、また半透明属性を計算するために、視線方向に沿った順に計算を行う必要がある。高速化を行うために様々な手法が提案されたが、現在では GPU（Graphic Processing Unit）のアーキテクチャにデータを載せて計算を行う、ハードウェアレンダリングが主流となっている。

構造工学や流体計算等、様々な分野で用いられる有限要素法の計算からは、不規則な格子構造を持つ不規則格子ボリュームデータが得られ、それに対するボリュームレンダリングを用いた可視化の要請も多い。例えば構造計算から得られた不規則格子ボリュームデータは複雑な内部形状を持つことが多く、断面図や等値面、境界面へのカラーマッピングでは不十分な場合がある。このような場合にボリュームレンダリングを用いることによって、全体の形状とスカラー値の分布を一度に可視化することができる。

しかし､不規則格子ボリュームデータ向けのボリュームレンダリング手法の開発は未だ不十分である。これまでに多くの手法が提案されたが、視線に沿った格子のソート処理が今後も開発を継続する必要のある重要な技術として位置づけられている。従来的なボリュームレンダリング技術は Sabella による密度発光モデル [2] から得られた輝度値方程式を計算することに帰着される。これを数値計算するためには、半透明な格子やサンプリング点を視線方向に対して順番に処理する必要があり、格子のソート処理や探索が必要とされる。従来手法ではこのソート処理は避けて通ることができず、大規模データをボリュームレンダリングする上でのボトルネックになっている。

しかし、密度発光モデルの起源である粒子モデルに注目することにより、ソート計算を必要としない効率のよいボリュームレンダリング手法が構築可能であることがわかった。粒子モデルとは不透明な自己発光する粒子がボリュームデータ内部に分布しているとする離散的なモデルであり、それを連続体とみなして計算を行うのが密度発光モデルである｡Sakamoto ら [3] はこの粒子モデルを用いたボリュームレンダリング手法として、粒子ベースボリュームレンダリング（Particle-based Volume Rendering, PBVR）を開発した。

Sakamoto らによる PBVR 法は不規則格子からなるボリュームデータを効率よく可視化することを期待されたが、この手法では伝達関数（ボリュームデータのスカラー値に対して色と不透明度をマッピングする関数）が与えられても、不透明度が定まらず、画像を一意に決定することができないという問題点があった。また、不規則格子に適用するための手法に関して、格子形状のラスタライズやボリュームデータの隣接情報を用いた格子の探索を必要とするものであり、十分に開発されているとは言えなかった [4]。

本論文では、画像を一意に決定するために粒子モデルを導入し、大規模な不規則格子ボリュームデータに適用するための粒子生成手法を提案する。そしてメモリ効率のよい画像生成手法を提案し、PBVR の画質に関しても議論する。

また、ボリュームレンダリング手法は一般的に急峻に変化する伝達関数が与えられた場合に画像にアーティファクトが生じ易い。その画質の改善について現在も様々な研究が行われているが、多くの手法が規則的な格子構造を持つボリュームデータにのみ適用可能な手法である。本論文では不規則格子向けの、急峻に変化する伝達関数に適用可能な、PBVR の粒子生成技術についても提案する。

以下では科学可視化の対象となるボリュームデータについて、そしてボリュームデータ向けの様々な可視化手法について概観する。特別に断わらない限り、論文中で用いる記号とその意味は、全て各章の中で定義される。

1.2　ボリュームデータ

ボリュームデータとは、実験や数値シミュレーションの結果として得られる、三次元空間中に離散的に定義された値の集合である。CT（Computational Tomography）スキャナや MRI（Magnetic Resonance Imaging）、PIV（Particle Image Velocimetry）等の測定技術からは、三次元空間中に規則正しく等間隔に、立方体上に並んだ計測点が得られる。一般的な用語として用いられるボリュームデータは、こうした直行等間隔に数値データが並んだものを指す場合が多い。

しかし、ボリュームデータは必ずしも直行等間隔に数値データが並んでいるとは限らず、例えば天候に関する情報のレーダー計測では、観測中心から放射状にデータが得られるため、極座標や球座標のように数値データが並んでいることもある。更に、欠損した医療データのように、規則正しく数値データが並んでいな

いデータも存在する。

そして、本研究の主要な対象である、人工的に作られた任意の形状を持つボリュームデータも存在する。これは有限要素法等の数値シミュレーションで計算されたとして出力されるボリュームデータである。このボリュームデータは、数値データに加えて、任意形状を現す格子を持つ。この格子とは、有限要素法において計算を行うための解析領域を複数の単純な形状の立体（要素）からなる部分領域に分割したものである。そして数値データは、節点と呼ばれる要素の頂点上に定義される。有限要素法では、数値データが節点ではなく要素の中心や稜線上に定義されることもあるが、IDW（Inverse Distance Weighted）法 [5] 等を用いて節点上の数値データを計算する事ができる。

以下では、ボリュームデータ上に定義される数値データの種類、そして格子構造について述べる。

1.2.1 数値データ

ボリュームデータに対する可視化手法は、数値データの種類によって異なる。可視化において数値データは、スカラー、ベクトル、テンソルの三種類に分類され、現在知られている殆の実験、数値シミュレーションから得られるボリュームデータ上の数値データはこの三種類のどれかに当てはまるか、もしくは変換可能である。これらの数値データは多くの場合節点上に離散的に定義され、補間関数を用いてボリュームデータ中の任意の位置に対する値が計算される。補間を併用することでスカラー、ベクトル、テンソルの各数値はボリュームデータ中の場を記述すると考えることができるため、フィールド値と呼ばれることもある。

スカラー値は大きさのみを持つ量で、スカラー場は座標変換によって変化しない場である。ボリュームデータに与えられるスカラー値はほとんどが実数のデータである。例えば、温度や圧力、密度等を挙げることができる。三次元空間の直交する軸をそれぞれ x、y、z 軸とする。可視化で扱うスカラーデータは、三次元空間からスカラー値 S への関数 $S(x,y,z)$のように記述される。

ベクトル値は大きさと方向を持つ量で、座標変換をした場合は適切に変化させる必要がある。ボリュームデータとして与えられるベクトル値の多くが三次元の幾何ベクトルである。例えば、流体の速度や加速度、構造物の変位等が挙げられる。ベクトルデータは、三次元空間からベクトル値 S_i（i=0,1,2）への関数 $S_i(x,y,z)$

のように記述される。

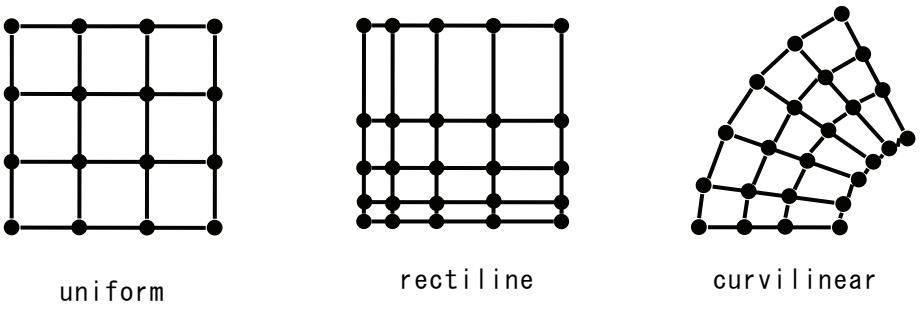
テンソル値は、広義には線形演算子を一般化したものであるが、工学的には多次元の配列で現される量である。配列の次元のことを階数と呼ぶ。0 階のテンソルはスカラー、1 階のテンソルはベクトルと等しい。2 階のテンソルデータ、は三次元空間からテンソル値 S_{ij}（i=0,1,2）（j=0,1,2）への関数 $S_{ij}(x,y,z)$のように記述される。ここで、添字の個数が階数を、添字のとりうる値の個数が次元を意味する。応用上最も多く用いられるのは 2 階のテンソルであり、構造解析から得られる応力テンソルやひずみ、拡散テンソル画像から得られる水の微分透磁率等が挙げられる。また、構造解析で用いられる弾性係数は 4 階のテンソルで記述されることがある。２階のテンソルは行列として表現され、固有値や固有ベクトルを計算して可視化に用いる事が多い。

1.2.2 格子形状

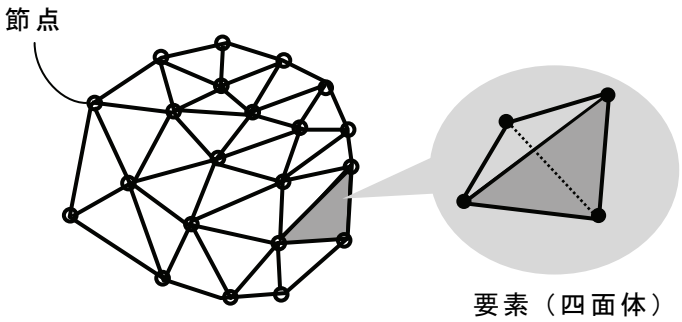
ボリュームデータはその格子形状によって、構造格子、不規則格子の二種類に分類される。節点が三次元空間中に何らかの規則性を持って並んだデータを構造格子と呼ぶ。構造格子では、各格子は規則的に並び隣接関係が自明である。逆に、隣接関係が自明でない格子を不規則格子と呼ぶ。図 1.1 (a) には二次元の場合の構造格子の例を、図 1.1 (b) には四面体からなる不規則格子の例を示す。

構造格子は更に、規則格子（uniform grid）、不等間隔直交格子（rectilinear grid）、湾曲格子（curvilinear grid）の三種類に分類される。以下では、三次元空間の直交する軸をそれぞれ x、y、z 軸とする。

規則格子は最も単純な格子形状である。三次元空間中に立方体が並んだ形状の格子であり、節点が直行等間隔に配置される。CT や MRI、PIV 等の実験計測装置、もしくは有限差分法や FDTD から得られるボリュームデータはこの規則格子ボリュームデータに分類される。このボリュームデータの節点上に定義される数値データのことをボクセルと呼ぶ。立方体の稜線は x、y、z 軸と並行とされる。普通、節点間の距離は全て 1 とし、格子の端点の一つを座標軸の原点上にあるとする。データの形式としては、始めに x、y、z 軸それぞれの方向における節点の個数（解像度と呼ばれる）が与えられ、後は数値データが何らかの規則に従って一列に並んでいるだけのことが多い。解像度と並びの規則が与えられれば、直ちに任意の座標を含む要素を取得することができる。



(a) 構造格子型ボリュームデータ



(b) 不規則格子型ボリュームデータ（四面体格子の場合）

図 1.1　ボリュームデータの形状

不等間隔直交格子は直方体から構成される格子である。規則格子と同様に、直方体の稜線が x、y、z 軸と並行になっているが、各軸方向に並ぶ節点が不等間隔になっている格子である。データの形式としては、規則格子ボリュームデータと同様の情報に加え、各軸における節点の座標を持つ。

湾曲格子は規則格子を歪ませた形状をしており、六面体から構成される。一般的に歪曲格子は写像法を用いて生成される。これは何らかの関数を用いて規則格子を写像して格子生成を行う手法である。例えば、極座標や円筒座標、球座標のような格子は湾曲格子として分類される。また、所望の領域の格子の分割数を増やした境界適合格子は、流体解析で用いられる事が多い。データの形式としては、規則格子ボリュームデータと同様の情報に加え、全節点の座標を持つ。任意の座標を含む要素を取得するのは、特別な場合を除いて簡単ではなく、規則格子と比べて計算時間が大きい。写像に用いた関数を推定して格子の探索を高速化する手法 [6] が提案されている。

不規則格子を構成する要素としては、四面体、四角錐、三角柱、六面体等、様々

な形状が用いられる。ほとんどの格子が単一種類の要素から構成されるが、複数種類の要素が混在した重合格子も存在する。先述したとおり、有限要素法の計算を行う上で任意形状を近似するため用いられる。計算の精度を高めるために、要素の稜線や面上、または要素の内部等に節点数を追加することもある。このような要素は適用する補間関数の次数が上がるため、高次要素と呼ばれる。この要素は節点の位置を調整し、適切な形状関数を与えることによって、曲面から構成された形状にすることもできる。データの形式としては、全節点の数値データと座標、そして節点のインデックスを用いた要素の情報から構成される。

1.2.3 隣接グラフ

格子を構成する要素の隣接関係を記述するデータ構造で、特にグラフを用いて表現されるものを隣接グラフ（Adjacency graph）と呼ぶ。可視化においてボリュームデータは、離散的な数値データに対して補間関数を作用させて、連続な場として扱われることが多い。このとき、格子内部の任意の位置を含む要素や隣接する要素を特定する必要がある。例えばレイキャスティングならば視線に、流線表示ならば流線に沿った要素の探索が必要とされる。

構造格子の要素間の隣接関係は全て規則格子と同様に自明であるため、隣接グラフが用いられることはない。ただし、任意の位置を含む要素の計算は、規則格子、不等間隔直交格子、湾曲格子の順に計算量が増す。

隣接グラフが用いられるのは不規則格子を扱う場合である。不規則格子ボリュームデータは格子のトポロジーを記述するデータとして、各要素を構成する頂点の接続情報しか持たないため、任意の座標を含む要素を見つけるために、要素の全探索が必要とされる。これ避けるために、前処理として要素の探索を行っておき、要素間の隣接グラフを生成する必要がある。この計算には、格子の頂点数や格子数に比例した時間が必要となる。もしも不規則格子が隣接グラフを持たなければ、

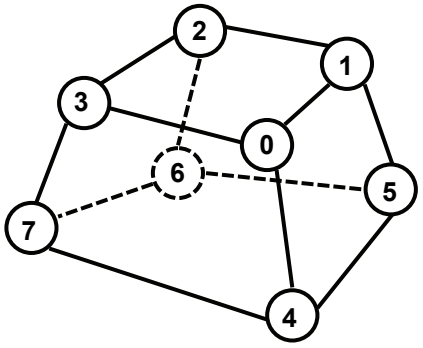
隣接グラフは、要素を構成する節点、辺、面、そして要素自身を用いて記述される（グラフ理論で用いられる点、辺、面と同一の意味ではないことに注意）。節点と節点の接続、辺と辺の接続等、用途によって様々な接続を記述するグラフが生成可能である。そしてコンピュータグラフィックスでは、要素間の接続を記述する隣接グラフが最も頻繁に用いられる。以下では不規則格子ボリュームデータ

に対する要素間の隣接グラフについて記述する。

隣接グラフを最も単純に実現する手段は、隣接行列を用いることである。しかしこの方法は、要素数の二乗の接続を持つためグラフが巨大化しやすく、小さなボリュームデータにしか適用できない。

要素と要素は同一の面を共有して接続している。そのため、面を介して要素の接続情報を記録しておけば、格子の探索を効率的に行うことができる。隣接情報は、節点や面、要素等に与えられたインデックスを用いて、各々の対応関係を表(今後テーブルと呼ぶ)として記述されている。このテーブルを作成するためにはまず、要素と節点、そして面に関する接続順番の定義を行う必要がある。

要素や面は、形状に一対一対応した節点のインデックスの並びで定義される。分野や用いるソフトウェアによって様々な接続順番の定義が用いられる。図 1.2 (a) に六面体要素を定義する節点の接続順番の例を示す。図 1.2 (b) に、要素の面の局所的なインデックスの定義の例を示す。



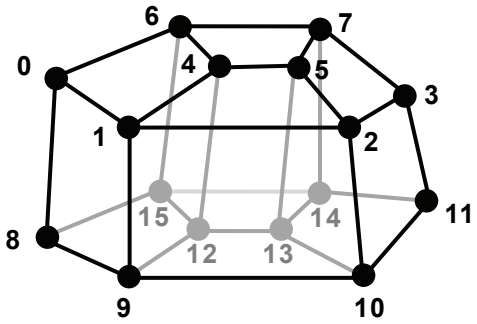
(a) 六面体要素に対する
頂点の接続順

face 0 = (0, 1, 2, 3)
face 1 = (7, 6, 5, 4)
face 2 = (0, 4, 5, 1)
face 3 = (1, 5, 6, 2)
face 4 = (2, 6, 7, 3)
face 5 = (0, 3, 7, 4)

(b) 頂点の接続による
局所面の構成

図 1.2　六面体要素

隣接情報のデータ構造は、節点テーブル、要素－節点テーブル、要素－局所面－要素テーブルから構成され、それぞれを参照し合いながら用いる。節点テーブルは節点座標とインデックスの対応を与える。要素－節点テーブルは、要素のインデックスと、その要素を構成する節点のインデックスの対応表である。要素－局所面－要素テーブルは、要素のインデックスと、その要素の面を通じて接する隣接要素のインデックスの対応表である。図 1.3 に不規則六面体格子と、それに対応する隣接情報の例を示す。図 1.3 (b) の 1 列目が要素のインデックス、2 列目が要素－節点テーブル、3 列目が要素－局所面－要素テーブルを現す。



（a）不規則六面体格子

Cell ID	Vertex Connection (Vertex ID)								Local face ID					
									0	1	2	3	4	5
0	0	1	4	6	8	9	12	14	(0)	(1)	3	1	(2)	(3)
1	4	5	7	6	12	13	15	14	(4)	(5)	3	2	(6)	0
2	5	2	3	7	13	10	11	15	(7)	(8)	3	(9)	(10)	1
3	1	2	5	4	9	10	13	12	(11)	(12)	(13)	2	1	0

（b）不規則六面体格子に対する隣接情報

Exterior face ID	0	1	2	3	4	5	6	7	8	9	10	11
Cell ID	0	0	0	0	1	1	1	2	2	2	2	3

（c）境界面－要素テーブル

図 1.3　不規則六面体格子と隣接情報

要素－局所面－要素テーブルは、要素の局所面に与えられたインデックスに対応して、隣接する要素のインデックスが与えられる。もしもある面に接する要素が存在しない場合、その面は格子の境界面を意味するので、その面に境界面のインデックスを与え、境界面テーブルに登録する。図 1.3 (b) の要素－面－要素テーブル中の括弧に囲まれた数字は、境界面のインデックスを意味する。得られた境界面のインデックスと要素のインデックスの対応関係は、図 1.3 (c) に示す、境界面－要素テーブルに記録する。この情報は次節で述べる、分割された不規則格子に関する隣接情報を生成するために用いられる。また、テーブル中の要素のインデックスと境界面のインデックスを区別するため、何らかの形でフラグを与えておく。例えば、フラグを格納しておくブーリアン型の配列を用いる、または、

境界面となるテーブル中の値を負数として定義する、などすることで判別が可能である。

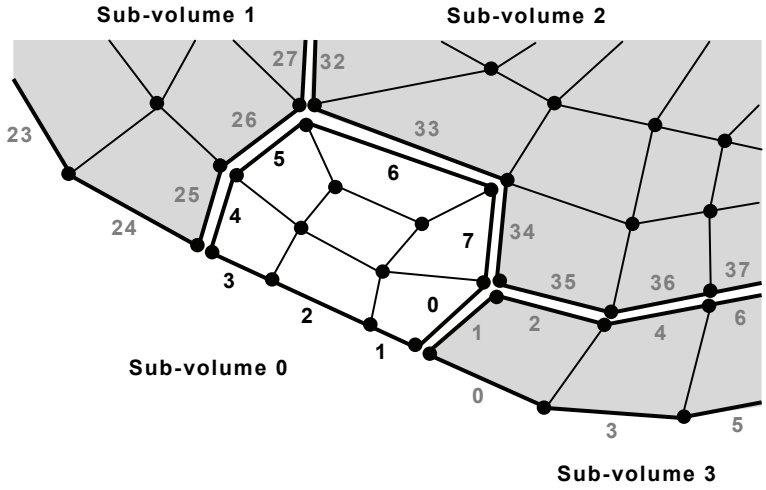
隣接情報を作成するために、ハッシュテーブルを利用する [7] 。要素を構成する面は、殆どの場合三角形か四角形である。三角形であっても四角形であってもそれを構成する節点の番号のうち大きいものから三つ使うことにより面を一意に識別することができる。隣接格子情報の作成にはこの三つの節点番号をキー・格子番号を値とするハッシュテーブルに登録する。格納時にハッシュテーブルを参照して、この三つの節点番号をもつ登録データが存在すると登録データに記録されている格子番号を取り出し、隣接格子の番号として格子データに追加する。

1.2.4 分割された不規則格子データ

高精度の計算や、複雑な形状を持つ解析領域を扱うために、有限要素法で用いられる格子の分割数が増加する傾向にある。それに伴い解析領域を複数に分割して、スーパーコンピュータや PC クラスタで計算を行う手法が開発された。その結果、ボリュームデータも複数領域に分割されて出力されるようになった。このような分割された格子の一つを部分ボリュームと呼ぶ。

部分ボリュームを超えて格子の探索を行うためには、各部分ボリュームの境界面を用いた、境界面間の対応関係を記録したテーブル、境界面－境界面テーブルを用いる。全ての部分ボリュームが、自身に隣接する部分ボリュームに関する境界面－境界面テーブルを持つ。図 1.4 (a) (b) には、複数の部分ボリュームと、ある部分ボリュームに関する境界面－境界面テーブルの例を示す。境界面－境界面テーブルは、境界面のインデックスに対して、接触する境界面部分ボリュームの境界面が存在するかどうかを示すフラグ、隣接する部分ボリュームのインデックス、そして隣接する部分ボリュームの境界面のインデックスが関連付けられている。

部分ボリューム間の隣接関係を計算するために、互いに接する境界面の組を特定する必要がある。ここで、部分ボリューム間で接触する境界面同士は、一致すると仮定する。境界面のマッチングを取るために、各境界面の重心を用いる。この計算を効率良く行うために、規則格子に重心位置を登録し高速化する手法等が提案されている [8] 。



(a) 部分ボリュームに分割された格子

Exterior face ID	Exterior flag	Adjacent sub-volume ID	Adjacent exterior face ID
0	0	3	1
1	1	-	-
2	1	-	-
3	1	-	-
4	0	1	25
5	0	1	26
6	0	2	33
7	0	2	34

(b) Sub-volume 0 の境界面－境界面テーブル

図 1.4　部分ボリュームと境界面－境界面テーブル

1.3　ボリュームデータ可視化手法

ボリュームデータを可視化するための手法は、その用途に応じて使い分けられる。例えばデータ全体を俯瞰したい場合と、見るべき部位や値が既に明確な場合では、用いられる可視化手法は異なる。

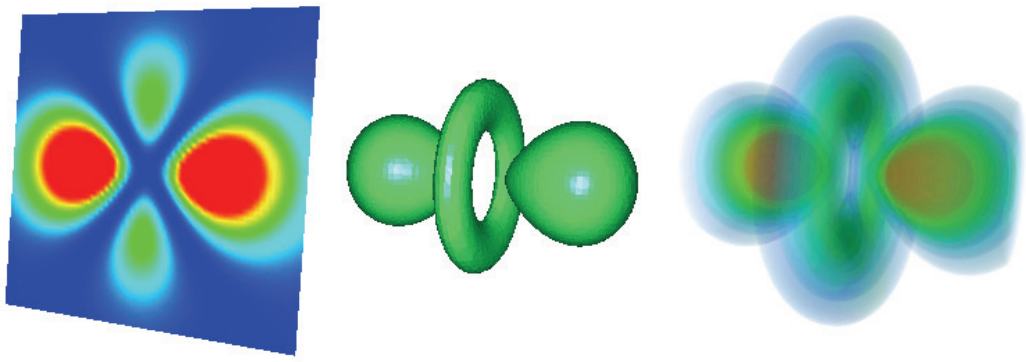
スカラー場が定義されたボリュームデータを可視化する手法は、断面コンター、等値面表示、ボリュームレンダリング、そして境界面のワイヤーフレーム表示やポリゴン表示等がある。

規則格子からなるボリュームデータの可視化には、実装が容易なため、断面コンターが広く用いられる。この手法はボリュームデータの任意の断面上のスカラー値の分布に対して色を割り当て、二次元でデータを表示する手法である。図 1.5 (a) に断面コンター表示の例を示す。

等値面とは、ある三次元空間のスカラー場の値を $S(x,y,z)$ としたとき、$S(x,y,z)=c$ (c は任意定数) を満たす点の集合であり、特殊な場合を除いて、空間中に曲面を形成する。この曲面はポリゴンの集合で近似表現される。ここでの c を閾値と呼び、この値をユーザーが自由に変えて等値面を生成することで、スカラー場の分布を幾何形状として視覚化する。図 1.5 (b) に等値面表示の例を示す。ボリュームデータから等値面を生成する手法としては Marching Cubes 法 [9] があげられる。Marching Cubes 法は等値点同士の接続テーブルをあらかじめ用意しておくことにより、等値面を高速に生成する手法である。

ボリュームレンダリングはボリュームデータ全体を半透明に描画する手法である。スカラー値に対する色と不透明度をユーザーが自由に指定し、ボリュームデータ内部のデータ分布の様子を視覚的に捉えることができる。図 1.5 (c) にボリュームレンダリングの例を示す。この技術の詳細は次章で述べる。

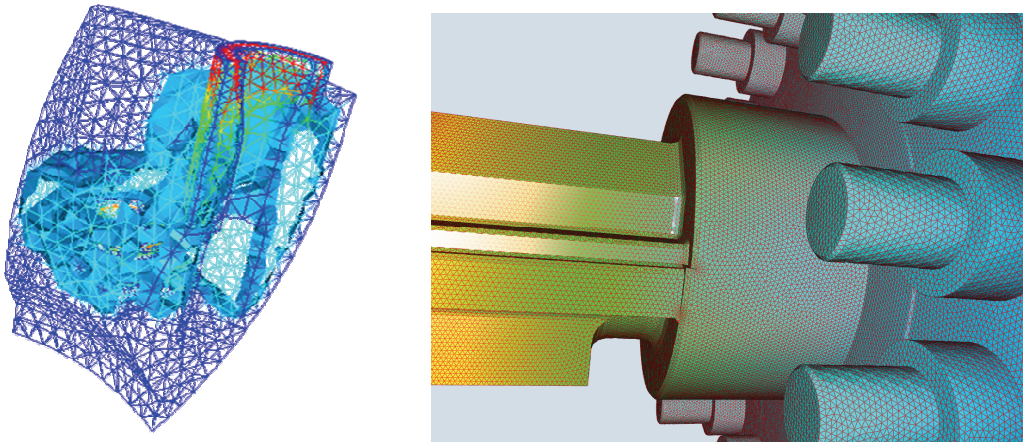
図 1.5 は同一のデータに対する、断面コンターと等値面、ボリュームレンダリングによる可視化の例である。これら三つの手法の中で最も情報量の多いものはボリュームレンダリングである。ボリュームデータを隙間なく描画できるため、スカラー値の分布の全容を把握することができる。また、より詳細な情報を得るために、あえて可視化する領域を制限することもある。ボリュームレンダリングの場合、興味のあるスカラー値の範囲の不透明度を上げ、他を下げることで実現される。更に、着目したい閾値や断面の位置等が分かっている場合、等値面や断面コンターを用いた可視化によって見たいものを強調することができる。



（a）断面コンタ　　（b）等値面　　（c）ボリュームレンダリング

図 1.5　水素原子の電荷密度分布を持つボリュームデータに対する可視化結果

不規則格子ボリュームデータの場合、解析領域の幾何形状そのものを見るために、境界面をワイヤーフレームやポリゴンを用いて表示する手法もしばしば用いられる。こうした場合は境界面上のスカラー値の分布に対して色を割り当て可視化する。図 1.6 (a) は、不規則格子ボリュームデータをワイヤーフレームと等値面を用いて表示した結果である。図 1.6 (b) は境界面をポリゴンとワイヤーフレームで表示した結果である。

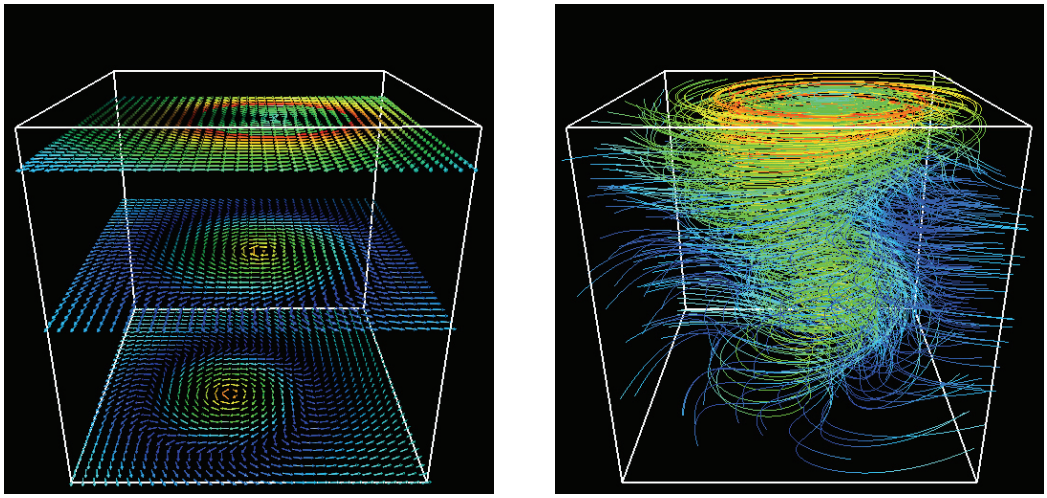


（a）ワイヤーフレーム　　（b）ポリゴン

図 1.6　不規則格子ボリュームデータの境界面の可視化

ベクトル場の可視化手法として、グリフや流線が用いられる。グリフとは、ベクトルの方向と大きさを、矢印の形状で表現する手法である。任意の位置におけるベクトルを知りたい場合に有効である。また流線とは、空間中のベクトルが接線ベクトルになるような曲線であり、電場の場合には電気力線、磁場の場合には

磁力線とも呼ばれる。流線を生成するためには、空間中に初期位置を与え、そこを開始点として数値積分を行う。図 1.7 (a) と (b) に、同一のデータに対するグリフ表示と流線表示の結果を示す。



（a）グリフ　　（b）流線

図 1.7　ベクトル場の可視化手法

テンソルボリュームの可視化では、テンソルデータをベクトルやスカラーに変換して、それぞれベクトル・スカラーボリュームデータ用可視化技術を使うことが多い。よく用いられるのは、テンソルデータに対して固有値分解を行い、その結果得られる固有値・固有ベクトルを利用することである。

テンソルグリフとは、すべての固有値が実数であるテンソルデータに対して、直交する三本の矢印で固有ベクトルと固有値を表現する手法である。矢印の方向は固有ベクトルの方向に平行で、矢印の長さでその大きさを表す。固有ベクトルは、方向に任意性があり、両方向に矢羽根を描画する。この場合、固有値の正負により内外向きに違いをもたせる。

超流線とは、固有ベクトルデータが接線ベクトルに平行となるような曲線であり、応力テンソルボリュームで最大固有値に対応する固有ベクトルに対して計算された超流線は、与えられた外力がどのように伝播するのかを理解する上で有用な可視化画像を提供する。固有ベクトルには方向の任意性があるので、格子内部で固有ベクトルの補間を行うときは、その方向を維持するよう計算上での注意が必要となる。また、医用画像計測装置の機能が高まり、手軽に拡散テンソル画像を撮影することができ、非侵襲で脳の神経線維を可視化することが可能となって

いる。神経線維を可視化するには、テンソルの最大固有値に対する固有ベクトルについて流線を描画することとなる。図 1.8 は、拡散テンソル画像から生成された脳神経線維を表す。

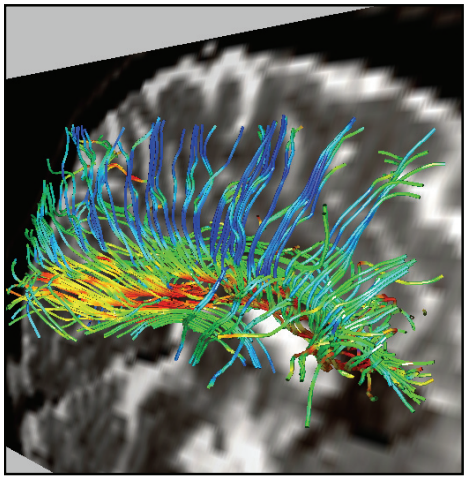


図 1.8　テンソル場の超流線

計測装置の機能が高まっても撮像の解像度がまだまだ粗く、一つの画素に百程度の神経線維が通過する程度なので、格子で神経線維が交差するケースをうまく処理できない場合がある。神経線維の方向がきちんとそろっている場合には格子において固有値の最大値が突出して大きいが、交差してしまう場合には、最大固有値とそのつぎの固有値の大きさにあまり差がない状態となる。このような場所を特定するには、最大固有値とそのつぎの固有値の大きさが等しくなってしまうような点がその格子に存在するかどうかを調べる。このような点を縮退点と呼ばれ、これを表示することでテンソル場の幾何学的な特徴を大きく捉えることができる。

1.4　論文の概要

本論文では、PBVR の画質に関する問題点を解決し、大規模な不規則格子ボリュームデータに適用するための粒子生成手法や、メモリ効率のよい画像生成手法を提案した。そして、提案手法を流体計算や構造解析の数値シミュレーションから得られた大規模な不規則格子ボリュームデータに適用し、有用性を示した。本論文は 7 章から構成されており、以下でその概要を述べる。

第 1 章では研究背景について述べ、可視化対象となるボリュームデータの種類や構成について整理した。そしてボリュームデータ向けの科学的可視化手法を概観し、ボリュームレンダリングの用途や位置づけを明確にした。

第 2 章では提案手法を理解するために必要な、ボリュームレンダリングの原理と用いられる技術、そして提案手法の要素手法である PBVR について述べた。ボリュームレンダリングにおいて重要な概念である粒子モデルが、関与媒質に対する光の散乱現象を取り扱うために開発されたことについて記述した。そして密度発光モデルからボリュームレンダリングの原理である輝度値方程式が導出される課程を記述した。提案手法の理解に必要な技術として、ボリュームデータに色と不透明度をマッピングする関数である、伝達関数について詳解した。また、計算を行う上で必要な三次元グラフィックスの基礎である、座標空間と陰影計算について述べた。提案手法の立ち位置を明確にするため、不規則格子ボリュームデータ向けの従来手法について述べた。そして、提案手法の要素技術である PBVR について述べた。

従来の PBVR には、伝達関数に対して画質が一意に定まらないという問題点があった。第 3 章では密度発光モデルから粒子密度推定式を導出し、生成粒子数を計算する手法を提案した。この手法により PBVR から生成される画像を一意に決めることが可能になった。また、PBVR で生成される画像の画質についても検証した。そして、ピクセルの分割数であるサブピクセルレベルによって画質を制御できることが明らかになった。提案手法を四面体要素からなる大規模な不規則格子に適用し、スケーラビリティを確認した。

従来の PBVR は四面体以外の不規則格子に対して粒子のサンプリングを行うことができなかった。第 4 章では粒子密度推定式を用いて、PBVR を不規則格子に適用するための粒子分布生成手法を提案した。また、画像重畳を用いて従来の PBVR よりもメモリ効率のよい画像生成手法を提案した。また、この手法により画像詳細度制御を実現した。提案手法を有限要素法で計算された実データに適用し、従来手法ではボリュームレンダリングが困難だった大規模不規則格子ボリュームデータ（領域分割された 7200 万の六面体要素や、2600 万の四面体二次要素から構成されたボリュームデータ）をボリュームレンダリングすることに成功した。

PBVR による画像は、粒子密度推定法により計算されるボリュームデータ内部の粒子密度分布に依存するため、適切な粒子生成が必要となる。しかし、伝達関

数が急峻に変化する場合、発生粒子数の計算と粒子生成を正確に行うことが困難であり、画像にアーティファクトを生じた。第 5 章では、四面体格子に適用可能な高品位な粒子生成手法である、四面体向けのレイヤードサンプリング法を提案した。そして第 6 章ではその技術を発展させ、六面体格子に適用可能なレイヤードサンプリング法を提案した。様々な不規則格子ボリュームデータに適用し、画質の検証を行った。その結果、従来の PBVR と比べてアーティファクトが少なくなっていることを確認した。

第 7 章では、結論として、各章で得られた成果を要約し、残された今後の課題について述べた。

第2章　ボリュームレンダリング技術

ディスプレイ上で三次元空間中に配置された不透明な物体を投影法で描画しようとする場合、透視投影であれ平行投影であれ、奥行き表現と引き換えに手前の物体がその奥にある物体を遮蔽する。不透明なオブジェクトは高速な描画が容易であり広範に用いられているが、内部を見透かし何らかの知見を得る目的で可視化を行ったのであれば、視線の遮蔽は観察者のストレスとなり、発見を阻害する可能性がある。単に内部を見ることが目的の場合、手前のオブジェクトを取り去ってしまえばよいが、その代わりに両者の相対的な位置関係に関する情報は失われる。空間的な情報を可能なかぎり維持したまま内部を見るためには、オブジェクトに半透明の属性を与えて描画することが有効である。

ボリュームレンダリングでは、与えられたボリュームデータにユーザー指定の伝達関数を適用し、スカラー値に対する色情報と不透明度の情報を割り当て、画像面に結果を表示する。Sabella による粒子発光モデル [2] が提案されて以後、計算機の高速化とボリュームレンダリング技術の発達により、伝達関数の変更や画像面上でのボリュームデータの回転・拡大などが対話的に操作できることが求められており、またボリュームデータも大規模・複雑化してきている。この章ではまず半透明物体の描画のための粒子モデルについて概観し、次に近年の不規則格子向けボリュームレンダリング技術について述べる。そして最後に、本研究の基本技術である粒子ベースボリュームレンダリングについて述べる。

2.1　関与媒質に対する画像の生成

空間的に連続に広がるボリュームデータを表現する手法は、煙や雲のような物体（関与媒質と呼ばれる）のアナロジーとして登場した。関与媒質の描画は初め、コンピュータグラフィックスの分野において研究がなされた。その第一歩は、Blinn がディスプレイ上に土星の輪を描くために粒子モデルを用いて光の散乱現象を記述したことである [10] 。そしてその手法は Kajiya の研究により、ボリュ

ームレンダリング方程式 [11] へと一般化された。この方程式を数値計算することでフォトリアリスティックな関与媒質の画像を得ることができた。

ここで特筆すべきは、Kajiya が提案した手法において、関与媒質の密度が定義されたスカラーボリュームデータを用いていたという点である。任意の密度ボリュームに対して半透明感を考慮した画像を生成できるこの技術は、やがて Sabella により、今日我々が知るボリュームレンダリングと言う手法に結実される。この説ではボリュームレンダリング法の礎となった Blinn による散乱モデルと、Kajiya によるボリュームレンダリング方程式を紹介する。

2.1.1　Blinn の散乱モデル

関与媒質を描画するための最初のコンピュータグラフィックスモデルの一つは Blinn によって提案された。彼は、光を反射する微小な氷の粒の集合から構成される土星の輪の画像を生成する手法を記述した。

Blinn は光の散乱と影、そして粒子群内部に伝播する光の移送に関するモデル化を行った。そのモデルにおいて、光線が粒子に対して一度だけ反射されることが仮定された。例えば、複数回の反射は無視される。関与媒質中で光線が粒子に対して反射を繰り返す現象のことを光の散乱と呼ぶ。この散乱現象の単純化は媒質中の粒子の反射率が小さいならば成り立つ。Blinn は更に、土星の輪を同一の半径を持つ粒子がある密度で一様に分布こと、そして分布する空間が薄い平面であることを仮定して計算を行った。

Blinn のモデルは、粒子群による光の散乱に起因する関与媒質の影、つまり光の遮蔽の効果も扱うことができる。またこのモデルは、粒子に遮られること無く雲の背後から来る光の総量として、雲の層の半透明感も扱うことができる。各ピクセルにおける雲の輝度値は、視線方向（視点からピクセルへ向かう直線の方向）の積分によって計算される。

雲画像生成では、視点・視線方向・光源位置を与え、光源からのエネルギーが雲の中で散乱・吸収を受けながら視点に到達する割合を計算する。Blinn は、視点と粒子、粒子と光源を結ぶ 2 本のパイプだけを考え、この中に他の粒子がひとつもないとき、散乱を受けた放射エネルギーが視点に到達するとして、視点に到達する放射強さ B（輝度値）を以下のように計算する（図 2.1）。

$$B = (w/\mu)\psi(\theta_a)\rho\pi r^2 \int_0^T P(0,V)dT' \tag{2.1}$$

ここで、

w :粒子表面反射率

ψ:位相関数（2 本のパイプにおいて放射エネルギーが伝達される割合）

θ_a:二つのパイプのなす角度

μ:法線ベクトルと視線ベクトルのなす角度の余弦

ρ:粒子密度（単位体積あたりの粒子の個数）

r :粒子の半径

T :平面層の厚み

T':平面層の厚み方向にとった座標軸

V :パイプの体積

$P(0,V)$:パイプに粒子が存在しない確率

を現す。

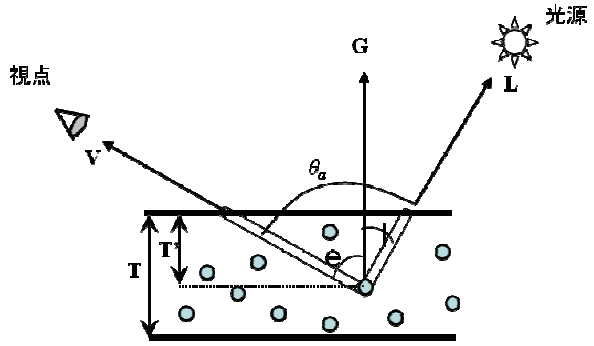


図 2.1　Blinn のモデル

画像化対象の平面層において、粒子がポアソン分布に従っていると仮定する。一般的に、ポアソン分布において、単位時間中に平均で τ 回発生する事象がちょうど k 回（k は 0 を含む自然数、k = 0, 1, 2, …）発生する確率は、次式で表される。

$$P(N = k) = \frac{e^{-\tau}\tau^k}{k!} \tag{2.2}$$

ここで、

e :ネイピア数 (e = 2.71828...)

$k!$: k の階乗

τ:正の実数

を現す。τは、所与の区間内で発生する事象の期待発生回数に等しい。

雲画像生成においては、単位時間をパイプの体積、そして事象の発生回数を粒子の個数と置き換えて、確率を計算する。式(2.2)は平均粒子数が τ の時 k 個の粒子が存在する確率を表すこととなる。$k = 0$ のとき、すなわちパイプに粒子が存在しない確率は、

$$P(0,V) = e^{-\tau} = e^{-\rho V} \tag{2.3}$$

となる。従って、粒子密度は単位体積あたりの粒子の個数であり、2 本のパイプの体積を乗じることにより平均粒子数は以下のように計算される。

$$\rho V = \rho\left(\frac{\pi r^2 T'}{\mu_0} + \frac{\pi r^2 T'}{\mu}\right) \tag{2.4}$$

ここで、μ_0は法線ベクトルと視線ベクトルのなす角度の余弦を表す。これを式(2.1)に代入して輝度値を計算すると以下のようになる。

$$\begin{aligned} B &= \frac{w}{\mu}\varphi(\theta_a)\rho\pi r^2 \int_0^T \exp\left(-n\left(\frac{\pi r^2 T'}{\mu_0} + \frac{\pi r^2 T'}{\mu}\right)\right) dT' \\ &= w\varphi(\theta_a)\frac{\mu_0}{\mu_0+\mu}\left\{1-\exp\left(-n\pi r^2 T\left(\frac{1}{\mu_0}+\frac{1}{\mu}\right)\right)\right\} \end{aligned} \tag{2.5}$$

粒子が多ければエネルギーの到達する割合が小さくなり、実際の距離ではなく粒子の数で光の通過距離を定義するほうがわかりやすい。このため平均粒子数τのことを光学的厚さとも呼ぶ。また、この光学的厚さに負符号をつけ、指数をとったもの、すなわち、式(2.3)を透明度と呼び、光の通過しやすさを表す。

2.1.2 ボリュームレンダリング方程式

Kajiya は Blinn のモデルを発展させ、高い反射率をもつ粒子に対する光の多重散乱を扱うモデルから光の散乱方程式を導出した。このモデルは雲や煙のような高い反射率を持つ粒子から構成される関与媒質に適している。そして散乱方程式を効率よく計算するために、粒子密度を定義したスカラーボリュームデータを用いて、レイトレーシングで描画する技術を提案した。この密度ボリュームの導入によって、Blinn が仮定した粒子密度が一定という制約を取り除き、数値シミュレーション等で得られた任意の密度分布に対する計算を行うことができる様になった。ただし Kajiya は散乱方程式を高速に計算するために、Blinn と同様の単一

散乱の仮定をおき、光源からの直接的な寄与のみを考慮に入れた。また粒子の自己発光による寄与も無視された。そしてこれらの近似を用いて、視線方向に対して漸進的に輝度値を計算する、レイトレーシングのアルゴリズムを提案した。

現在では関与媒質を描画するために、Kajiya の散乱方程式に媒質の自己発光による寄与の項を加えた、ボリュームレンダリング方程式が用いられている。自己発光を考慮することで雲や煙のみならず、炎のような媒質の描画も可能になった。この方程式は媒質中のあらゆる場所からあらゆる方向に射出される輝度値を計算する積分方程式である。

ボリュームレンダリング方程式には、視線方向や輝度値の放射方向、入射方向等、様々な方向が出現する。方向ωは、ボリュームデータが存在する空間に対して球面座標が与えられれば、2 つの偏角θとϕを用いて以下のように計算される。

$$\omega = \sin\theta\cos\phi e_x + \sin\theta\sin\phi e_y + \cos\theta e_z \tag{2.6}$$

ここで e_x 、e_y 、e_z は空間の直交基底となるベクトルである。輝度値 B はその位置 x だけでなく、輝度値の放射する方向ωにも依存する 5 変数の関数であり、$B(x,\omega)$のように書かれる。また方向にω'に関して、半径が 1 となる単位球面Ωにおける面積分の計算も行われる。この積分値の計算は、下式のように、球面座標系における積分に変換するほうが便利である。

$$\int_\Omega f(\theta,\phi)d\omega' = \int_0^{2\pi}\int_0^{\pi} f(\theta,\phi)\sin\theta\, d\theta\, d\phi \tag{2.7}$$

ボリュームレンダリング方程式は、方向ωに対する輝度値 B の変化に対して記述される。この値は方向微分係数を用いて以下のように現される。

$$\omega\cdot\mathrm{grad}\,B(x,\omega) \tag{2.8}$$

ここで grad B は x に関する勾配である。

方向ωに対する吸収による輝度値の損失は下式で与えられる。

$$\omega\cdot\mathrm{grad}B(x,\omega) = -\pi r^2\rho(x)B(x,\omega) \tag{2.9}$$

ここで r は粒子半径、ρ は粒子密度（単位体積当たりの粒子数）である。コンピュータグラフィックスの分野では、$\pi r^2\rho$ が吸収係数として与えられることも多い。

粒子の自己発光による輝度値の増加は下式で与えられる。

$$\omega\cdot\mathrm{grad}B(x,\omega) = \pi r^2\rho(x)c(x,\omega) \tag{2.10}$$

ここで c は粒子自身が放つ単位面積当たりの輝度値である。

位置 x への散乱による輝度値の変化は下式で与えられる。

$$\omega \cdot \mathrm{grad}\, B(x,\omega) = \sigma_s(x)\frac{1}{4\pi}\int_\Omega p(x,\omega',\omega)B'(x,\omega')d\omega' \tag{2.11}$$

ここでσ_sは散乱係数、p は位相関数、B'は様々な方向からの入射輝度、ω'はその入射方向である。散乱係数は散乱が起こる確率を意味し、粒子密度と密接に関係する値である。例えば密度が 0 の領域に対して散乱は起こらず、高密度の領域では散乱が起こりやすい。位相関数とは散乱光の分布、つまり w'から入射して w に抜ける光線の量を表現するす関数である。位相関数は単位球面上で積分した結果が 1 になるよう正規化されている。

$$\frac{1}{4\pi}\int_\Omega p(x,\omega',\omega)d\omega' = 1 \tag{2.12}$$

式(2.9)、(2.10)、(2.11)の結果をまとめると、方向ωに対する輝度値の変化は下式で現される。

$$\omega \cdot \mathrm{grad}\, B(x,\omega) = \\ \left(-B(x,\omega)+c(x,\omega)\right)\pi r^2\rho(x)+\sigma_s(x)\frac{1}{4\pi}\int_\Omega p(x,\omega',\omega)B'(x,\omega')d\omega' \tag{2.13}$$

視線方向について計算するために、位置 x を下式のように置く。

$$x = t\mathbf{n}+\mathbf{r}_0 \tag{2.14}$$

ここで $\mathbf{r}_0$ は視点の位置、$\mathbf{n}$ は視線の方向であり大きさが 1 である。当然ω=$\mathbf{n}$ である。t は視線に沿った位置を表現するパラメータである。式(2.14)を(2.13)に代入して下式を得る。

$$\frac{dB(t)}{dt} = \left(-B(t)+c(t)\right)\pi r^2\rho(t)+\sigma_s(t)\frac{1}{4\pi}\int_\Omega p(t,\omega')B'(t,\omega')d\omega' \tag{2.15}$$

ここで各関数の引数について、例えば $B(t\mathbf{n}+\mathbf{r}_0,\mathbf{n})$ を $B(t)$ としたように、定数となる部分を省略して変数だけで表記した。

式(2.15) は両辺に以下の積分因子を乗じることにより解くことができる。

$$\exp\left(\int_{t_n}^{t}\pi r^2\rho(u)du\right) \tag{2.16}$$

区間 $[t_n, t_0]$ で積分することで、微分方程式は次のように解かれる。

$$B = \int_{t_n}^{t_0} c(t)\pi r^2\rho(t)\exp\left(-\int_t^{t_0}\pi r^2\rho(\lambda)d\lambda\right)dt + \\ \int_{t_n}^{t_0}\frac{\sigma_s(t)}{4\pi}\left(\int_\Omega p(t,\omega')B'(t,\omega')d\omega'\right)\exp\left(-\int_t^{t_0}\pi r^2\rho(\lambda)d\lambda\right)dt \tag{2.17}$$

ここで t_0 と t_n はボリュームデータと視線との交点のうち、視点から最も近い点および遠い点を表す。式(2.17)がボリュームレンダリング方程式であり、関与媒質を描画するために解かなくてはならない方程式である。

2.2　粒子発光モデル

Kajiya の手法を継承した Sabella は、スカラーボリュームデータを可視化するという目的で、現在の科学可視化におけるボリュームレンダリングに用いられている、輝度値方程式を導出した。この手法は元の光学モデルにおいて、粒子を反射体ではなく光の放射体と見なすことによって得られたものである。つまり、ボリュームレンダリング方程式における周囲からの光の散乱による放射輝度の増加を粒子自身の発行現象と読み替えて単純化を行ったのである。リアリスティックな画像の生成と異なり、ボリュームデータの可視化という目的にあっては、ボリュームレンダリング方程式における光の散乱という現象は必ずしも有用なものではなく、むしろ内部に対する可視性を妨げる原因にもなっていた。加えてこの手法は、光の散乱という極めて計算コストの高い項を無視することによって、ボリュームレンダリングという手法の計算速度を大幅に向上させた。

Sabella の提案したモデルは密度発光モデルと呼ばれる。このモデルでは、散乱のない不透明な自己発光する粒子がボリュームデータ中に分布するという仮定がおかれた。粒子の大きさは画素の大きさに比べて小さいとされた。このモデルでは、粒子密度は単位体積における粒子の数と決められた。ボリュームレンダリング画像はボリュームデータを通して画素から視点に届く輝度値を画素の色に設定して作成される。ここで視線上に、中心軸が視線に平行で断面積が A であるような円柱を考える。視線に沿って t 軸をとり、t_0 と t_n は、t 軸とボリュームデータとの交点のうち、視点に近い点と遠い点を意味するものとする。粒子密度関数を $\rho(t)$ と表すと、中心が t に置かれ、長さが dt であるような微小円筒は、$A \cdot dt$ の体積と、$N = \rho \cdot A \cdot dt$ 個の粒子を持つ。粒子の半径がすべて r の球であるならその投影面積は πr^2 となる。もし、それらがすべて光量 $c(t)$ で拡散的に輝くならば円柱の両底面間の輝度値の差は以下のように記述される（図 2.2)。

$$dB(t) \cdot A = (-B(t) + c(t)) \cdot \pi r^2 \cdot \rho \cdot A \cdot dt \tag{2.18}$$

従って次のような微分方程式が定式化される:

$$\frac{dB(t)}{dt} = \big(-B(t) + c(t)\big) \cdot \pi r^2 \rho(t) \tag{2.19}$$

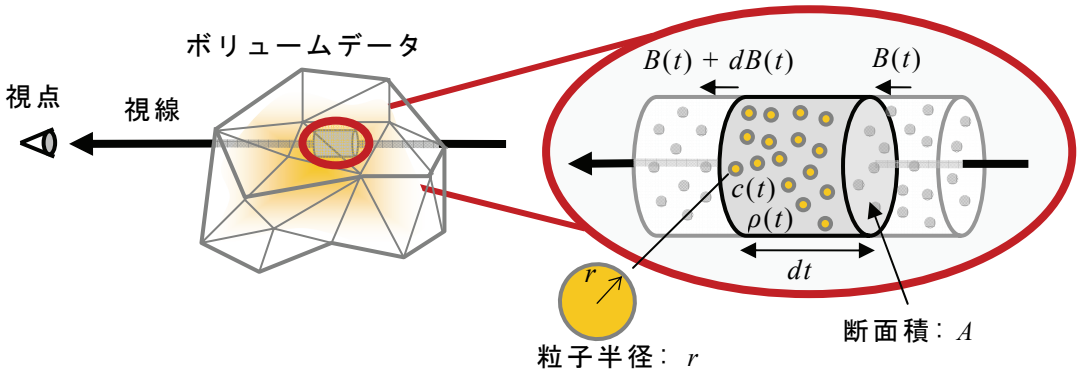


図 2.2　密度発光モデル

式(2.19) は両辺に式(2.16)の積分因子を乗じることにより解くことができる。区間 $[t_n, t_0]$ で積分することで、微分方程式は次のように解かれる。

$$B = \int_{t_n}^{t_0} c(t)\pi r^2 \rho(t) \exp\left(-\int_t^{t_0} \pi r^2 \rho(\lambda)\, d\lambda\right) dt \tag{2.20}$$

ここで t_0 と t_n はボリュームデータと視線との交点のうち、視点から最も近い点および遠い点を表す。

式(2.20)は輝度値方程式と呼ばれる、多くのボリュームレンダリング技術の基礎となっている方程式である。ここで、$B_0 = B(t_0)$ とし、指数項は粒子発光が視点に到達する確率を表す。

輝度値方程式を数値的に計算するために、粒子発光・粒子密度を一定値と見なせる長さ Δt （レイセグメントと呼ぶ）で視線上の積分領域を n 等分し、k 番目における粒子発光・粒子密度を一定値 c_k ・α_k とおく。このとき、輝度値方程式は以下のように計算される。

$$\begin{aligned} B &= \sum_{k=1}^{n} \int_{t_k}^{t_{k-1}} c_k \pi r^2 \rho(t) \exp\left(-\int_t^{t_0} \pi r^2 \rho(\lambda) d\lambda\right) dt \\ &= \sum_{k=1}^{n} \prod_{i=0}^{k-2} \exp\left(-\int_{t_{i+1}}^{t_i} \pi r^2 \rho(\lambda) d\lambda\right) \times \int_{t_k}^{t_{k-1}} c_k \pi r^2 \rho(t) \exp\left(-\int_t^{t_{k-1}} \pi r^2 \rho(\lambda) d\lambda\right) dt \\ &= \sum_{k=1}^{n} \prod_{i=0}^{k-2} \exp\left(-\int_{t_{i+1}}^{t_i} \pi r^2 \rho(\lambda) d\lambda\right) \times c_k \left(1 - \exp\left(-\int_{t_k}^{t_{k-1}} \pi r^2 \rho(\lambda) d\lambda\right) dt\right) \end{aligned} \tag{2.21}$$

k 番目の積分領域における指数項は、レイセグメントが十分に小さければ、式(2.3)の透明度と同等であり 0 から 1 までの値を取る。そこで 1 から透明度を引いた値を不透明度 α_k とし、以下のように定義する。

$$\alpha_k = 1 - \exp\left(-\int_{t_k}^{t_{k-1}} \pi r^2 \rho(\lambda) d\lambda\right) \tag{2.22}$$

この不透明度は視線方向に対して粒子発光が遮蔽される確率を表している。レイ

セグメントの長さは $\Delta t = t_{k-1} - t_k$ なので式(2.22)は以下のように簡単化される。

$$\alpha_k = 1 - \exp(-\pi r^2 \rho_k \Delta t) \tag{2.23}$$

この関係を使って、輝度値は次のように計算される。

$$B = \sum_{i=1}^{n} c_i \times \left(\alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j) \right) \tag{2.24}$$

通常、この不透明度 α は、ユーザーが指定する伝達関数において、スカラー値 S から変換される。また、スカラー値 S は位置 (x, y, z) の関数であるので、以下のように現すことができる。

$$\alpha = \alpha(S(x,y,z)) \tag{2.25}$$

もし、積分区間の長さがあらかじめ定めた Δt と異なる値 $\Delta t'$ となる場合は、α について、以下のような補正を行う必要がある。

$$\begin{aligned} \alpha_k &= 1 - \exp(-\pi r^2 \rho_k \Delta t) \\ \alpha'_k &= 1 - \exp(-\pi r^2 \rho_k \Delta t') \\ \therefore \alpha'_k &= 1 - (1 - \alpha_k)^{\frac{\Delta t'}{\Delta t}} \end{aligned} \tag{2.26}$$

この補正は、サンプリング間隔を適応的に変える場合に必要となる。

2.2.1　レイキャスティング

式(2.24)の輝度値計算を用いた代表的なボリュームレンダリング手法がレイキャスティングである。レイキャスティング法は視線上に設定されたサンプリング点ごとに評価された発光量（色成分）および不透明度を使って輝度値を計算する手法である [12] 。一般に、ボリュームレンダリングでは、入力となるボリュームデータの数値データは、スカラーデータとして表現されている。そのため、輝度値を計算するためには、伝達関数を利用して、スカラーデータを色成分と不透明度に変換する。

このときのスカラー値は、サンプリング点を含む格子の節点で定義されたスカラー値から補間計算を行うことによって求められる。視線上に設定されたサンプリング点を含む格子を特定するのは多くの計算時間を要する処理となる。多くのレイキャスティング法では視線とボリュームデータとの交点をまず計算し、そのうちで視点から一番遠いものを出発点として隣接格子情報を参照しながら、視線に沿って視点に向かって格子を次々と探索する。このような探索処理によりサンプル点が視線からの距離に遠い順から計算されていくことになり、式(2.24)は以下

のような漸化式で表わすことができる。

$$B_{k-1} = c_k \alpha_k + (1-\alpha_k) B_k \tag{2.27}$$

この漸化式を用いることにより効率のよい輝度値計算が可能となる。視点から向かって背面から積み重ねるように輝度値を計算することより back-to-front アルゴリズムと呼ばれる。また、このようにもとの色と新しい色をある比率 $(1-\alpha : \alpha)$ で混合することをアルファブレンディングと言う。このとき、最終的な輝度値 B は $B = B_0$ として計算できる。

一方、これとは逆に視線とボリュームデータとの交点のうちで視点から一番近いものを出発点として隣接格子情報を参照しながら、視線に沿って視点に向かって格子を次々と探索する方法も提案されている。このような視線探索処理ではサンプル点が視線からの距離に近い順から計算されていくことになり、式(2.24)は以下のような漸化式で表わすことができる

$$\begin{aligned} B^{k+1} &= c_{k+1}\alpha_{k+1}A^k + B^k \\ A^{k+1} &= (1-\alpha_{k+1})A^k \end{aligned} \tag{2.28}$$

ただし、A は累積不透明度を表す。このとき、累積不透明度 A が 1.0（不透明）となるときは、それより奥に位置するサンプリング点で発せられる光が視点に到達しないため、計算を打ち切る（early ray termination）ことが可能となり、その時点での色成分が最終輝度値 B となる。視点から向かって前面から積み重ねるように輝度値を計算することより front-to-back アルゴリズムと呼ばれる。この計算打ち切りの可能性が back-to-front アルゴリズムに対する優位性となる。

2.3　伝達関数

ボリュームレンダリングでは、スカラー値に対応した色と不透明度をユーザーが任意に与える。スカラー値 s を色 c に対応付ける関数を色関数 $c(s)$、不透明度 α に対応付けるための関数を不透明度関数 $\alpha(s)$ といい、これらを合わせて伝達関数（Transfer function）と呼ぶ。伝達関数の調整は可視化結果に大きな影響を与える操作であり、ボリュームデータの特徴やデータ内部で生じる様々な変化を適切に描画するための重要な技術の一つである。

ユーザーは自身の目的に合わせて伝達関数を自由に設定する。例えば伝達関数を適切に与えることで、特定領域の強調や分類を行うことができる。医用分野で

の CT や MRI から構成されるボリュームデータを対象にした場合、人体の各組織に対応した応答値があらかじめ分かっていることが多い。これを利用して、骨を表すスカラー値の領域に対して白、皮膚に対して肌色、血管を表す領域に対して赤のように色を割り当てることによって実物に近い可視化結果を得ることができる。そして骨を表す領域に対しては不透明度を高くし、その他の部分は不透明度を低くすることで、骨部分の形状を明確にした可視化結果を得ることができる。また､例えばボリュームデータ全体に渡るスカラー値の分布を可視化したい場合、不透明度を低い値で一定にすることで、スカラー値の分布を色の分布として可視化することができる。ここに紹介した他にも様々な伝達関数の設計方法が存在するが、最適な伝達関数を自動的に決定する方法は未だ開発されておらず、現在でも重要な研究対象である。

伝達関数を用いたスカラー値のマッピングの例を図 2.3 に示す。図中央に示す伝達関数は、横軸がスカラー値、縦軸が色、もしくは不透明度の強度になっており、ユーザーはこの平面上に色関数と不透明度関数を与える。この例では色の指定に、R（赤）・G（緑）・B（青）の 3 成分から構成される、RGB 色空間が用いられている。この他に、色相（Hue）・彩度（Saturation）・明度（Value）の 3 成分から構成される､HSV 色空間がしばしば用いられる。図 2.4 に HSV 色空間を用いた伝達関数のインターフェースの例を示す。このインターフェースの最上段にある“Color Palette”は横軸が彩度、縦軸が明度になっており、Color Palette の右にある縦に細い帯が色相になっている。

伝達関数の編集を行うには、図 2.3 に示すようなスカラー値と不透明度（もしくは色）に関する領域の中に､自由曲線か制御点を用いて関数の形状を描画する。スカラー値に対する一価関数のみが関数の形状として用いられる。自由曲線を用いる場合、スカラー値を細かい階調に分割し、離散的なテーブルとして曲線を保存する。制御点を用いる場合は、ユーザーがスカラー値と関数値の組を与えて、制御点間の関数値は補間計算されることが多い。また、制御点としてガウス関数の平均と分散をしていし､そのガウス関数の和として関数を構成することもある。

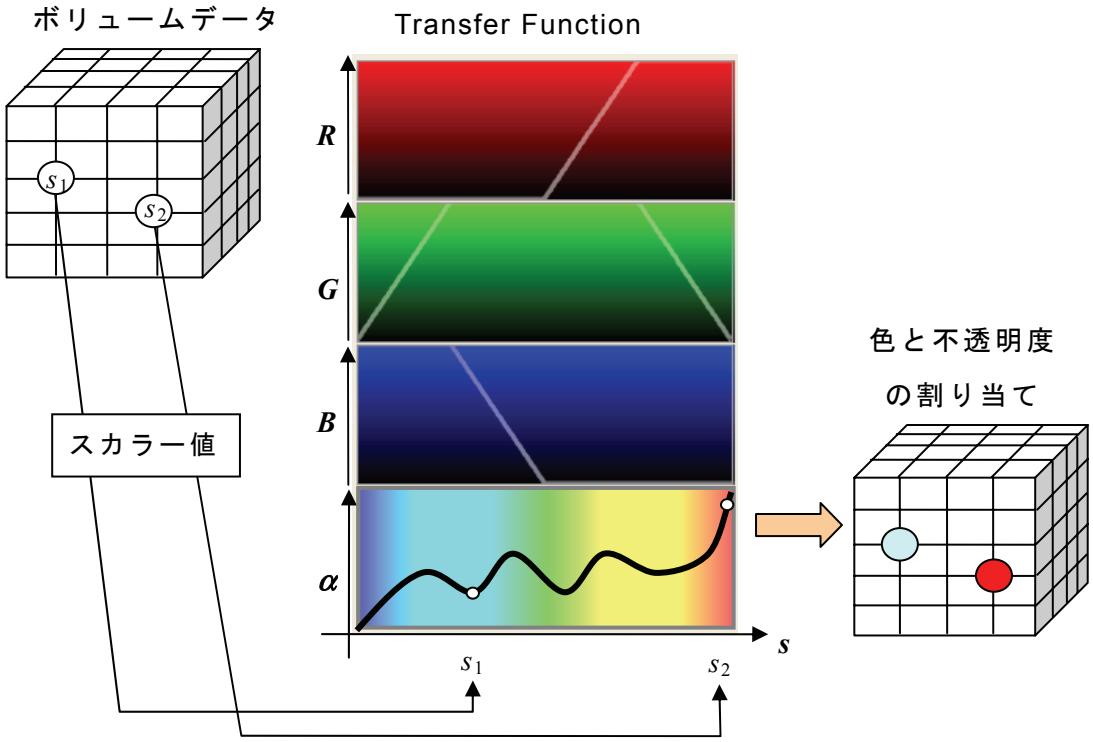


図 2.3　伝達関数とスカラー値の対応

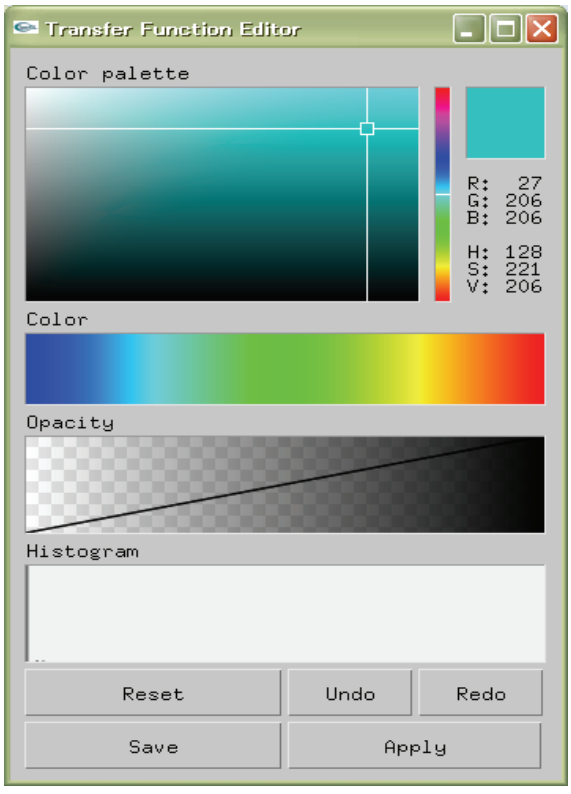


図 2.4　HSV 色空間を用いた伝達関数のインターフェース

2.4　座標空間

三次元空間中にある物体を画像面上に描画するためには、物体の位置、カメラの位置、視野角等の様々なパラメータと、それらが存在する空間の座標を適切に管理する必要がある。スクリーンはピクセルに対して計算を行うために二次元の座標空間を持ち、カメラは三次元空間中の物体を捉えて画像面に投影する必要がある。画像面やカメラの座標に加えてボリュームデータは、それが計測された空間や数値シミュレーション時に設定された物理空間に関して定義されている。このような複数ある座標空間を破綻無く扱うために、五つの座標空間とそれらに対する座標変換が定義されている [13] 。五つの座標系を以下に列挙する。

- オブジェクト座標系（Object coordinate system）
- 世界座標系（World coordinate system）
- カメラ座標系（Camera coordinate system）
- 投影座標系（Projection coordinate system）
- ウィンドウ座標系（Window coordinate system）

オブジェクト座標系とは描画対象となる物体が持つ座標系であり、$\mathbf{x}_o=(x_o,y_o,z_o)^T$ で記述する。ここで $*^T$ は $*$ の転置を表す記号である。物体の実寸値やシミュレーション計算において都合の良い座標値が定義されている。通常、オブジェクト座標系は物体ごとに異なるため、統一空間内に複数の物体を表示する場合には、相対的な位置関係を考慮し、適切に表示する必要がある。ちなみに、本研究においてボリュームデータが存在する空間を大域座標系と呼ぶことがあるが、これはオブジェクト座標系と同一のものである。この呼び方は有限要素法で使用されるもので、本論文では特に局所座標系（格子の要素中の計算空間のこと）と対比させる場合に用いる。

世界座標系は、三次元空間中に物体を配置するときに基準となる座標系であり、$\mathbf{x}_w=(x_w,y_w,z_w)^T$ で記述する。複数の物体を適切に表示するためには、各々独立に定義されたオブジェクト座標系を世界座標系に収めて管理する。つまり、世界座標系を、統一空間を記述する座標系として定義し、その座標系上に物体を設置する。

カメラ座標系は、世界座標系上に配置された物体を描画する際の視点（カメラ）を基準とした座標系であり、$\mathbf{x}_c=(x_c,y_c,z_c)^T$ で記述する。このカメラ座標系は、視線方向が z_c 軸の負の方向を向き、y_c 軸の正の方向が上、x_c 軸の正の方向が右を向くように定義する。このように定義することで、投影面上の横軸と縦軸が、カメラ座標系の x_c 軸と y_c 軸の方向に一致する。

実際の投影の計算では、画像面に入る物体だけを描けばよい。このとき描かれる領域は視点を頂点として無限に伸びる四角錐になる。また、処理を効率化するために、画像面に対して最も遠方にある面を定義し、その面と画像面との間に有る物体だけを描く。ここで視点に近い画像面のことを前方クリップ面、遠い面のことを後方クリップ面と呼ぶ。これらクリップ面で切り取られた四角錐の領域は四角錐台を形成し、その領域のことを視錐台と呼ぶ。投影座標系は、カメラ座標系に定義された視錐台を[-1,1]x[-1,1]x[-1,1]の範囲に収めた座標系であり、$\mathbf{x}_{proj}=(x_{proj},y_{proj},z_{proj})^T$ で記述する。

ウィンドウ座標系は、画像面上に定義されている座標系であり、$\mathbf{x}_{win}=(x_{win},y_{win})^T$ で記述する。この座標系では、一般的には、画像面の左隅が原点となり、右方向に x_{win} 軸の正の方向を、上方向に y_{win} 軸の正の方向をとる。

これらの座標空間に対する座標変換は四つあり、描画対象とする物体を以下の手順で変換し、最終的に画像面上に描画する。

- モデリング変換：オブジェクト座標系から世界座標系への変換
- ビューイング変換：世界座標系からカメラ座標系への変換
- 投影変換：カメラ座標系から投影座標系への変換
- ビューポート変換：投影座標系からウィンドウ座標系への変換

これらの変換は、同次座標（Homogeneous coordinates）に対する三次元の幾何学的な座標変換として表現される。同次座標とは、三次元の座標(x,y,z)を 0 でない実数 w を用いて(wx,wy,wz,w)と現す座標である。以下、同次座標に対する通常の座標のことを一般の座標と呼ぶ。例えば同次座標が(X,Y,Z,W)となっている場合、一般の座標では$\left(\frac{X}{W},\frac{Y}{W},\frac{Z}{W}\right)$と現される。$w$ の値は任意であるが、普通は w=1として、一般の座標(x,y,z)に対応する同次座標を$(x,y,z,1)$と書く。以下、特に断りのない限り、ベクトルの第一成分を x 成分、第二・三・四成分を y・z・w 成分と

呼ぶ。

同次座標を用いることで、平行移動や拡大・縮小、回転のような幾何学的な変換を 4x4 行列で表現することができる。そしてそれらを、同次座標で現された列ベクトルの左側に適切な順序で掛け合わせていくことにより、一連の変換が実行される [13]。

変換行列は、変換後の座標系の基底ベクトル $\mathbf{E}=\langle \mathbf{e}_x,\mathbf{e}_y,\mathbf{e}_z \rangle$ を考慮して構成される。ここで、この変換にはベクトルの「成分」に対する変換と、座標系の「基底」に対する変換の二種類があることに注意する必要がある。成分に対する変換とは、座標系の基底ベクトルを変化させること無く、ベクトルの成分だけを変化させることである。成分への変換 **A** は下式で与えられる。

$$\mathbf{A}=\begin{bmatrix} \mathbf{e}_x & \mathbf{e}_y & \mathbf{e}_z & \mathbf{0} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.29}$$

ここで $\mathbf{0}=(0,0,0)^{\mathrm{T}}$ である。**A** を、物体を指し示す位置ベクトルに掛けることで、座標系を変えること無く物体を動かすことができる。それに対して、基底に対する変換とは、位置ベクトルを動かすのではなく、その位置ベクトルを指定するための座標系の基底ベクトルを変化させる行列である。基底に対する変換 **A'**は下式で与えられる。

$$\mathbf{A}'=\mathbf{A}^{-1}=\begin{bmatrix} \mathbf{e}_x^{\mathrm{T}} & 0 \\ \mathbf{e}_y^{\mathrm{T}} & 0 \\ \mathbf{e}_z^{\mathrm{T}} & 0 \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.30}$$

ここで $\mathbf{0}=(0,0,0)$である。この変換は、カメラ位置を定義するビューイング変換で用いられる。

2.4.1 モデリング変換

モデリング変換は、物体に対するスケーリング（拡大・縮小）、回転、平行移動に関する行列を用いて、物体のオブジェクト座標を世界座標系内部の所望の範囲へ収める計算をする。つまり、モデリング変換の行列を **M** とすると、$\mathbf{x}_{\mathrm{w}}=\mathbf{M}\mathbf{x}_{\mathrm{o}}$ という計算を行う。この行列はベクトルの「成分」に対する変換である。

原点を中心として x_{o}、y_{o}、z_{o} 軸方向にそれぞれ s_x 、s_y 、s_z 倍変化させる変換は、下式のスケーリング行列 $S(s_x,s_y,s_z)$を用いて記述することができる。

$$S(s_x,s_y,s_z)=\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.31}$$

x_o、y_o、z_o 軸方向にそれぞれ t_x 、t_y 、t_z だけ平行移動させる変換は、下式の平行移動行列 $T(t_x,t_y,t_z)$を用いて記述することができる。

$$T(t_x,t_y,t_z)=\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.32}$$

回転を表す変換は回転軸ごとに個別に考える。ある点を x_o 軸まわりにθ_x だけ回転させる変換は、回転行列 R_x を用いて記述される。同様に、y_o 軸まわりにθ_y だけ回転させる変換は回転行列 R_y を用いて、z_o 軸周りにθ_z だけ回転させる変換は回転行列 R_z を用いて記述される。これらの回転行列を下式に示す。

$$R_x=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.33}$$

$$R_y=\begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.34}$$

$$R_z=\begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.35}$$

ある点を、これらの回転行列を合成した回転行列 $R_z R_y R_x$ によって回転させたとき、回転前の点の位置と回転後の点の位置の間の関係を与える（θ_x, θ_y, θ_z）をオイラー角と呼ぶ。

2.4.2 ビューイング変換

ビューイング変換は、世界座標系のカメラ位置（視点）$\mathbf{c}=(c_x,c_y,c_z)^{\mathrm{T}}$ に関する平行移動と、カメラの向きに関する正規直交基底を合成して得られる。カメラの見る位置（物体が一つの場合、その重心位置）を **g** とする。この時カメラの光軸（視線方向）を現すベクトル **f** は以下のように計算される。

$$\mathbf{f} = \frac{\mathbf{g}-\mathbf{c}}{|\mathbf{g}-\mathbf{c}|} \tag{2.36}$$

カメラの上方向を定義する大きさ 1 のベクトルを **u** とする。光軸ベクトルが z_w 軸の負の方向を向くような正規直交基底 $\mathbf{E}_c$ は、上方向ベクトルと光軸ベクトルを用いた外積により下式のように計算される。

$$\mathbf{E}_c = \langle \mathbf{u}\times(-\mathbf{f}), -\mathbf{f}\times(\mathbf{u}\times(-\mathbf{f})), -\mathbf{f} \rangle \tag{2.37}$$

カメラを動かすということは、物体を動かすのではなく、座標系の基底ベクトルを変化させることに等しい。$\mathbf{E}_c$ を用いてビューイング変換行列 **V** は下式で定義される。

$$\mathbf{V} = \begin{bmatrix} (\mathbf{f}\times\mathbf{u})^{\mathrm{T}} & -c_x \\ ((\mathbf{f}\times\mathbf{u})\times\mathbf{f})^{\mathrm{T}} & -c_y \\ -\mathbf{f}^{\mathrm{T}} & -c_z \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.38}$$

ここで **0**=(0,0,0)である。カメラ座標は $\mathbf{x}_c=\mathbf{V}\mathbf{x}_w$ で計算される。

2.4.3　投影変換

カメラ座標系から投影座標系への変換を表す投影変換は、透視投影（Perspective projection）と平行投影（Orthogonal projection）の 2 種類の投影方法がある。透視投影は空間中のすべての点が 1 つの視点に収束する投影方法である。透視投影では、近くの物体は大きく、遠くの物体は小さく描画することで遠近感を出すことができ、実際に人の目で見た空間に近い描画が可能である。ここでは透視投影を用いた投影変換について紹介する。

透視投影は物体から放射された光のうち、投影面を通過し視点位置に集まる光だけが投影面上で結像すると考える、ピンホールカメラのような投影モデルである。任意の位置から放たれた光の投影面上の位置は、それらの座標を用いて簡単に計算することができる。原点に視点を置く場合、任意の点の座標をその z 成分で割り算することで、z=1 平面上の座標が計算できる。投影面が $z=n$ で表される場合、点(x,y,z)を投影した投影面上の座標(x',y')は下式で計算される。

$$\begin{cases} x' = n\dfrac{x}{z} \\ y' = n\dfrac{y}{z} \end{cases} \tag{2.39}$$

同次座標を用いることで、$z=n$ 平面への投影計算を記述することができる。同

次座標においてw成分の値がzであるとき、一般の座標のx成分とy成分は式(2.39)のようにzで除算される。投影変換ではこの同次座標の性質を利用した変換行列を用いる。

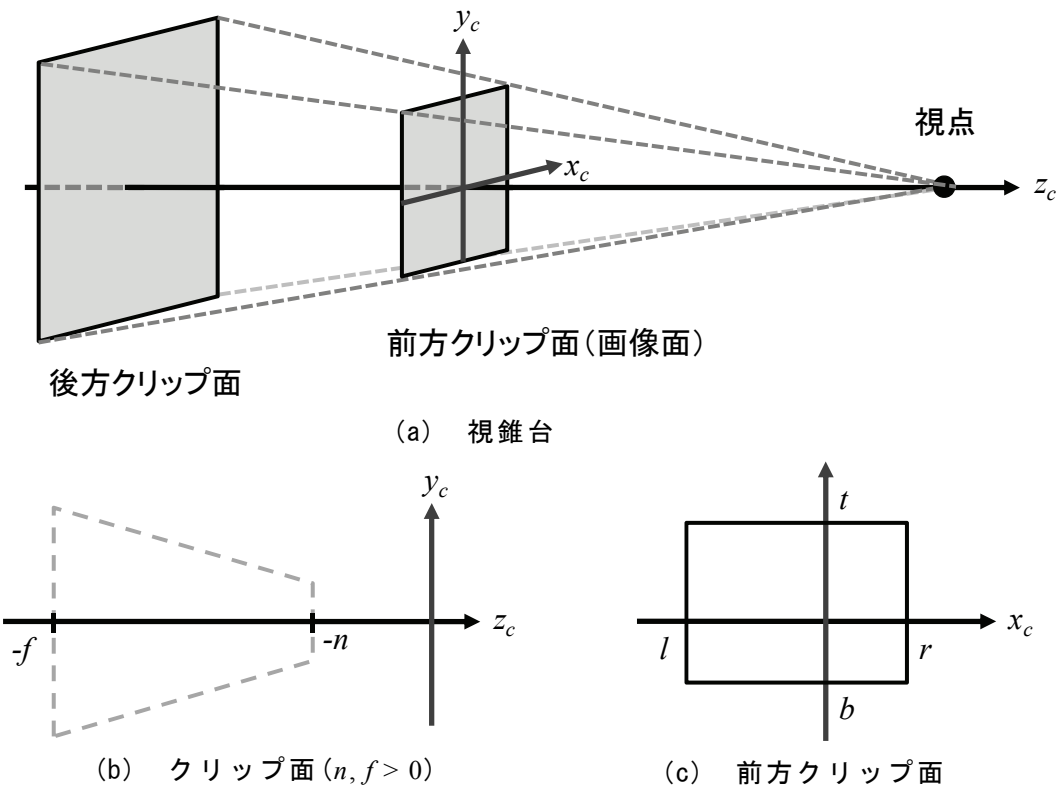


図 2.5　クリップ面と視錐台

図 2.5(a)に示すように、透視投影は視点と画像面の端を結んだ線で囲まれる領域を描画する。そして、無限に遠くまで描画することを避けるために、画像面（前方クリップ面）とそれに並行な後方クリップ面を考え、両面に挟まれた領域のみを描画する。この四角錐台の領域のことを視錐台と呼ぶ。視錐台の視点の位置は、ビューイング変換によって原点に置かれ、光軸はz_c軸と一致する。図 2.5(b)に示すように、前方クリップ面はz_c=－n、後方クリップ面はz_c=－fの平面で定義される。光軸が視点（原点）の方向を向いているため、両クリップ面の座標は負の値を取る。また、図 2.5(c)に示すように、前方クリップ面は x_c-y_c 平面上での領域[l,r]x[b,t]で定義される。

投影変換は視錐台で定義される領域[l,r]x[b,t]x[n,f]を領域$[-1,1]^3$ に変換する処理である。投影変換は以下の三つの処理から構成される。

2.4.3.1　左手系への変換

これまで扱ってきた座標系は全て右手系であったが、ウィンドウ座標系を扱うためには左手系が扱いやすい。そのため $S(1,1,-1)$を作用させることで左手系へ変換する。変換後の座標を $\mathbf{x}_l=(x_l,y_l,z_l)$とする。この変換により、負の値をとっていた前方クリップ面と後方クリップ面の位置はそれぞれ正の値 n、f となる。

2.4.3.2　同次座標による除算を考慮した変換

次は前方クリップ面 $z_l=n$ への投影を計算するための、z 成分による除算を考慮した変換を行う。ここで同時に z_l の範囲を[-1,1]に正規化する計算も行う。このときの変換行列は以下のように現される。

$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.40}$$

$$A=\frac{f+n}{f-n}, B=\frac{-2fn}{f-n} \tag{2.41}$$

この行列による変換後の値は以下のように計算される。変換後の座標を $\mathbf{x}_p=(x_p,y_p,z_p)$とする。$\mathbf{x}_l$ の同次座標$(x_l,y_l,z_l,1)^{\mathrm{T}}$ に変換行列を作用させ w 成分で除算すると、一般の座標における x_p と y_p は以下の式で現される。

$$\begin{cases} x_p=n\dfrac{x_l}{z_l} \\ y_p=n\dfrac{y_l}{z_l} \end{cases} \tag{2.42}$$

また一般の座標における z_p は以下の式で現される。

$$z_p=\frac{f+n-\dfrac{2fn}{z_l}}{f-n} \tag{2.43}$$

式(2.43)に $z_l=[n,f]$を代入すると、z_p=[-1,1]へ変換されることが確認できる。実際の同次座標の w 成分による除算は、投影変換に関わる三つの処理全てが終了した後に実行される。しかしそれは行列に対してスカラー値を掛けることと同じであり、除算のタイミングによらず同じ結果が得られる。

2.4.3.3　投影位置の正規化

最後に前方クリップ面の領域$[l,r]$x$[b,t]$を領域$[-1,1]^2$ に変換する。この変換はスケーリング行列と平行移動行列を用いて以下のように現される。

$$S\left(\frac{2}{r-l},\frac{2}{t-b},1\right)T\left(-\frac{r+l}{2},-\frac{t+b}{2},0\right) \tag{2.44}$$

以上の三つの変換により領域$[l,r]$x$[b,t]$x$[n,f]$は領域$[-1,1]^3$ に変換される。これらの変換をまとめると以下の変換行列 **P** が得られる。

$$\mathbf{P}=\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{2.45}$$

この変換で得られる投影座標系は、正規化デバイス座標系とも呼ばれる。投影座標は $\mathbf{x}_{proj}=\mathbf{P}\mathbf{x}_c$ で計算される。

2.4.4　ビューポート変換

投影変換で得られた投影座標をビューポート変換することで、ウィンドウ座標上の値、つまり画像面上での位置が計算される。ウィンドウ座標における投影面の左下隅の位置を(p_x,p_y)、投影面の幅を *width*、高さを *height* とする。このときビューポート変換は以下のように現される。

$$\begin{aligned} x_{win} &= \frac{width}{2}x_{proj}+p_x+\frac{width}{2} \\ y_{win} &= \frac{height}{2}y_{proj}+p_y+\frac{height}{2} \end{aligned} \tag{2.46}$$

通常、*width* と *height* の比率は投影面の高さと幅の比率と等しくなるように決める。

2.5　陰影計算

照明を設定し、陰影処理（シェーディング）を施すことによって、物体に影をつけることができ立体感が増し、効果的に物体を表現することができる。シェーディングは、物体を構成する任意の面または頂点に対し、その位置、向き、材質および照明となる光源の位置、向きなどを考慮して物体表面で起こる光の反射をモデル化し、物体の色や明るさを決定する処理である。

ボリュームレンダリングは、入射光からの寄与を無視した密度発光モデルに立脚するため、画像に陰影が生じない。そこで、本来ポリゴンデータのような面を構成する要素に対する処理であるシェーディングを、ボリュームレンダリングの計算に用いる各サンプリング点の輝度値に対する減衰項として作用させることで、ボリュームレンダリング画像に陰影の効果を与える。

2.5.1 照明モデル

単純な照明モデルは、物体の表面から反射する光の強度を局所的に近似するものである。これは、局所的照明モデル（Local illumination model）と呼ばれ、物理的な正確さには欠けるが、実装が容易であり低い計算コストで効果的に立体感のある画像生成が可能であるため、コンピュータグラフィックス分野で幅広く利用されている。

このモデルでは、物体表面で起こる光の反射は、環境光、拡散反射、鏡面反射の組み合わせにより表現され、その強度（輝度）を計算することによって物体の色や明るさを決定することができる。

2.5.1.1 環境光

現実空間では、光源から放射された光は、まわりの環境によって複雑に散乱し、物体の影の部分であってもある程度その色や形を見ることができる。このような光は、環境光（ambient light）と呼ばれ、空間全体をほのかに照らし出す効果を与える。この環境光を厳密に計算するためには膨大な計算が必要とされるが、局所照明モデルでは、光源の向きに関係なく物体表面を均等に照明する光として、反射強度 $I_{ambient}$ は以下のように計算される。

$$I_{ambient} = k_a \cdot M_a \cdot I_a \tag{2.47}$$

ただし、I_a は入射光の強度（環境光の色）、M_a は物体の材質（環境光に対する物体色の成分毎の反射率）である。k_a は 0 から 1 の範囲で表される反射強度の調整を行うための定数である。

2.5.1.2 拡散反射

物体表面に入射した光は、その表面の微小な凹凸や表面下散乱の影響により、乱反射する。このような反射は、拡散反射（diffuse reflection）と呼ばれ、光の入

射角に関係なく全方向に均等に散乱する。そのため、拡散反射による反射強度 $I_{diffuse}$ は見る角度や視点位置に依存せず、以下のようにして計算することができる。

$$I_{diffuse} = k_d \cdot M_d \cdot I_d \cdot \cos\theta \tag{2.48}$$

ただし、I_d は入射光の強度（拡散光の色）、M_d は物体の材質（拡散光に対する物体色の成分毎の反射率）である。k_d は 0 から 1 の範囲で表される反射強度の調整を行うための定数である。また、θ は物体表面上の法線方向に対する光の入射角を表している。入射角 θ が 0 になるとき、つまり、光源が法線ベクトル方向にあるとき、反射強度は最大となり、入射角 θ が $\pi/2$ 以上となるときは、光源が物体の裏面を照らすことになるため反射強度は 0 になる。

実際には、物体表面上の法線ベクトル N、光源の方向を示す光源ベクトル L として、さらに N と L が単位ベクトルである場合は、式(2.42)は内積を計算することにより以下のように書き直すことができる。

$$I_{diffuse} = k_d \cdot M_d \cdot I_d \cdot (N \bullet L) \tag{2.49}$$

拡散反射を計算するサンプルコードを以下に示す。ただし、先に述べたように式(2.48)に示す θ が $\pi/2$ 以上のときは反射強度が 0 になることから、式(2.49)内の N と L の内積が負のときは拡散反射が 0 となることに注意する。

2.5.1.3　鏡面反射

物体表面が金属や鏡などのように滑らかな面である場合、光源方向から入射した光はある特定方向に反射する。このような反射は、鏡面反射（secular reflection）と呼ばれ、この反射による反射強度 $I_{specular}$ は、光源の位置と視点に依存し、見る方向により値が異なる。また、物体表面に光が反射し明るくなる部分（ハイライト）が生じ、光沢感の高い表現となる。物体表面における反射強度 $I_{specular}$ は、以下のように表すことができる。

$$I_{specular} = k_s \cdot M_s \cdot I_s \cdot \cos^n \varphi \tag{2.50}$$

ただし、I_s は入射光の強度（鏡面光の色）、M_s は物体の材質（鏡面光に対する物体色の成分毎の反射率）であり、φ は光の入射角に対する正反射ベクトル R と視線ベクトル V とのなす角を表している。k_s は 0 から 1 の範囲で表される反射強度の調整を行うための定数である。また、n は鏡面反射のハイライト特性を表す変数であり、値を大きくするにしたがい、鋭く集中したハイライト効果となる。

実際には、式(2.50)は R と V の内積を利用し以下のようにして計算される。

$$I_{specular} = k_s \cdot M_s \cdot I_s \cdot (R \bullet V)^n \tag{2.51}$$

また、正反射ベクトル R は光源ベクトル L と法線ベクトル N を用いて以下のように表される。

$$R = 2 \cdot (N \bullet L) \cdot N - L \tag{2.52}$$

鏡面反射を計算するサンプルコードを以下に示す。ただし、ここでも拡散反射の場合と同様に R と V の内積が負にならないように注意する。

2.5.2　シェーディング

2.5.2.1　ランバートシェーディング

鏡面反射成分を考慮せず環境光による反射と拡散反射のみから反射強度を計算することにより光沢感をなくしたシェーディングが可能となる。このようなシェーディング処理はランバートシェーディングと呼ばれ、以下のようにして反射強度を計算することができる。

$$I = I_{ambient} + I_{diffuse} \tag{2.53}$$

2.5.2.2　フォンシェーディング

環境光による反射および拡散反射に加え、鏡面反射も考慮して反射強度を計算することによって光沢感のあるシェーディング処理が可能である。このようなシェーディング処理は、フォンシェーディングと呼ばれ、以下のようにして反射強度が計算される。

$$I = I_{ambient} + I_{diffuse} + I_{specular} \tag{2.54}$$

2.6　不規則格子向けボリュームレンダリング手法

可視化技術が研究対象になって以来、多くのボリュームデータに関する可視化手法が開発されてきたが、そのほとんどが規則格子を前提とした技術である。そのため、不規則格子を規則格子に変換するという手法が現在も使われている。十分に細かい規則格子を用いて変換を行えば、元のデータの持っていた分布をある程度再現できるが、それはあくまでも近似であり、やはり元の不規則格子のまま

取り扱うことが望ましい。

この章では不規則格子向けのボリュームレンダリング手法について記述する。まず代表的な手法である、格子投影法、その精度を高める事前積分法、そしてポイントベースの手法であるスプラッティング法について述べる。次に様々な従来手法について概観し、不規則格子向けのボリュームレンダリング手法が持っている問題点を明確にする。

2.6.1 格子投影法

ボリュームレンダリング法では、レイキャスティング法のような画素から伸びる視線を基準としたアプローチと、ボリュームデータを構成する格子そのもの投影して画像生成を行うアプローチの２種類が知られている。前者を画素ベース、後者をオブジェクトベースのアプローチと言う。

四面体格子に対するオブジェクトベースの手法として、格子投影法が知られている。この手法は四面体要素を三角形で表現し、視点からの距離にしたがってこの三角形を並び替え、back-to-front アルゴリズムまたは front-to-back アルゴリズムを適用する [14] 。視点から見て最も厚みのある点で四面体の厚みを計算し、その点で、三つか四つの四面体に分け、さらにそれぞれを三角形として表現する（図 2.6）。この場合、分割して新たに定義された三角形では最も厚みのある点を共有しており、この頂点では、厚みの中心で補間された粒子密度と厚みを使って、不透明度を計算する。

$$\alpha = 1 - \exp\left(\pi r^2 \Delta t \frac{\rho_f + \rho_b}{2} \right) \tag{2.55}$$

三角形のその他の頂点では厚みがゼロであるために、不透明度はゼロとする。また、全ての頂点ではスカラーデータから色データに変換しておき、色・不透明度付三角形として描画する。

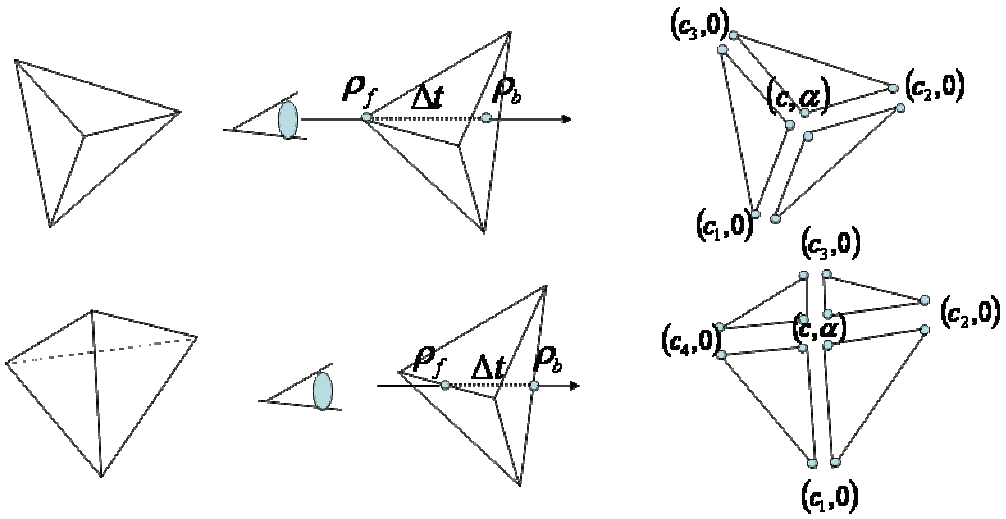


図 2.6　格子投影法

2.6.2 事前積分法

格子投影法では格子を表現する三角形の頂点で色・不透明度を評価し、三角形領域においてはこの頂点データを線形補間する。本来なら線形分布していない色・不透明度を線形補間することにより画質低下が生じる。このため、例えば、一様なスカラーデータを与えた場合に見えるはずのない格子の形が見えてしまう。また、四面体内部で不透明度が急峻に変化していても、最も厚みのある点で不透明度を平均的に計算しているので、この不透明度の急峻な変化を表現する画像を生成することが困難である。この問題を解決するために、伝達関数が決定された段階で、事前に精度の高い積分計算を行っておくという手法が事前積分法[15]である。格子投影法と同様に、視線と四面体格子との交点のうち、視線に近い方のスカラーデータ S_f と遠い方のスカラーデータ S_b の間でスカラーデータは線形補間される。格子投影法との違いは、スカラーデータの関数である粒子密度を単純に線形補間せず、その急峻な変化を考慮することができる点である。

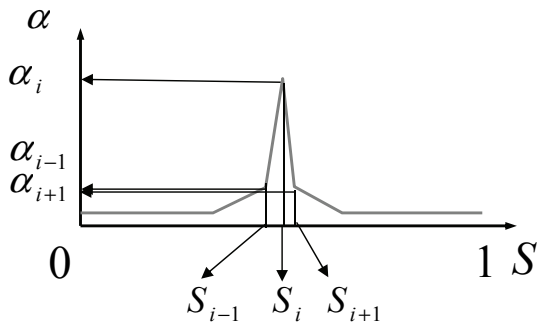


図 2.7　区分線形伝達関数

色・不透明度については、それぞれ、式(2.21)における積分区間内・外の計算を様々な区間を想定して事前に行っておく。式(2.21)では区間は視線上のパラメータ t を使って記述されているが、事前積分法では、視線に近い方のスカラーデータ S_f と遠い方のスカラーデータ S_b、そして区間幅 Δt を使ってこの区間を表現する。伝達関数を離散的なスカラーデータに対して表現し、隣接する離散スカラーデータ間では、線形補間を使って伝達関数値（色・不透明度）を計算する。すなわち、事前積分法では、積分区間において、伝達関数を区分的線形関数として表現する（図 2.7)。この仮定のもと想定する区間として隣接する二つの離散スカラーデータと区間の幅を使って色・不透明度を積分計算し、その結果を三次元テクスチャに登録しておく（図 2.8)。 三次元テクスチャは、三つのパラメータからデータを取り出せるようにした三次元配列で、登録時には離散的パラメータで行うが、取り出し時には連続的パラメータにより、三次元線形関数を使ってデータ値の補間を行う。

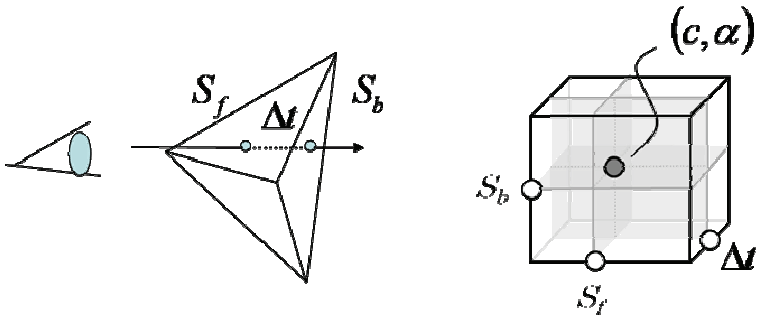


図 2.8　事前積分法

格子投影法と違って、格子そのものを三角形で表現するのではなく、格子の格子面（三角形）を利用し、この格子面が視点からの距離に応じてソートされていることが前提となる。格子面を大雑把にソートするには格子面の重心と視点まで

の距離を使ってソート計算を行うことになるが、面積の大きな格子面と小さな格子面が近接しているときに誤りが生じる場合がある。格子面処理において計算したスカラー値・区間幅をそのまま使うのでなく、一時的に保持しておき、数回の格子面処理を経て格子面の順序が決まってから、三次元テクスチャから色・不透明度を取り出すようにする手法 [16] が提案されている。

2.6.3 様々な不規則格子向けボリュームレンダリング手法

Koyamada [17] は、画素ベース、すなわち視線探索による不規則格子向けボリュームレンダリング法を提案した。視線探索法は、高品位の画像を計算することができる。しかしながらこの手法は、画素毎に計算される視線と格子との交差判定及び､不規則格子内部の視線方向の積分に高い計算コストが必要とされたため、高速なレンダリングや大規模なデータへの適用は困難であった。 Garrity [18] も同様の手法を提案した。

Koyamada ら [19] は､これら二つの部分における計算負荷を緩和するために、四面体格子の性質をうまく利用して、交差判定回数を減らしたり、輝度値計算を高速化したりする新しい格子探索手法の開発を行った。

Shirley ら [14] が四面体投影（Projected tetrahedral, PT）法を提案してから、不規則格子向けボリュームレンダリング手法の主流は、画素ベースから格子ベースへと移った。彼らの手法では、各四面体格子は視線に沿って奥行き順に画像面上に投影され､アルファ合成された結果､半透明画像が計算される。Williams [20] の研究成果をきっかけに四面体格子のソートについての研究成果が多く生み出された。Lucas [21] は不規則格子の格子面に関する投影によるボリュームレンダリング法を提案した。この手法では、格子面について奥行きソートを行った後、各面を画像面上で画素展開し、各面頂点での色・不透明度・奥行き値を線形補間により計算する。この手法では四面体以外の格子についても同様に処理することができるという特長がある。

Koyamada ら [19] は、三角形で構成される視線に垂直な曲面群を重ね合わせるとこによりボリュームレンダリングを実現するアルゴリズムを提案した。これら曲面群は視点を中心とする同心球スライス面を構成する。さらに､Koyamada ら [22] は、これらのスライス面を効率よく計算するために隣接する同心球面間における類似性に着目した手法の提案を行った。

Meredith と Ma [23]は不規則格子を八分木構造で近似し、それらをハードウェアサポートのあるスプラッティング法で描画する手法の開発を行った。スプラッティング法において、様々な種類の形状やアスペクト比を持つ不規則格子に対する正確なカーネルの構成は簡単ではない。そのためこの手法では、八分木の各格子に含まれる複数の要素の頂点の値を一つのカーネルで近似してスプラッティング法を行っている。また、八分木を視線に沿って後から前に探索するときは、八分木に含まれるあるひとつのノードにおいて、葉ノードのソートの有無は最終画像にほとんど影響を及ぼさないことが明らかにされた。

Wylie ら [24] はハードウェアサポートによる四面体投影法の高速化を実現したが、その時点でのハードウェアの性能からは高々49 万個の四面体が処理可能とされた（四面体ソートの計算コストは省く）。Roettger ら [25] は、四面体ソートに必要とされるメモリバンド幅が処理速度の上限を決定することを示し、ソート不要のボリュームレンダリング手法の提案を行った。彼らのレンダリングモデル粒子発光モデルから吸収効果を無視したものなので、その適用は炎、霧のような現象の表現に限定される。Csébfalvi [26] [27] もソート不要のボリュームレンダリング法を提案したが、先ほど述べた手法とは逆で、吸収効果のみを考慮しており、いわゆる X 線撮影像と同様の画像しか作成できない。

Callahan ら [16] は四面体投影法のソーティングの速度を向上させるため、統合的ソート手法（Hardware-assisted visibility sorting、HAVS）を提案した。この手法は、まずオブジェクト空間において、格子面重心を用いた大雑把なソートを行い、その後、画像空間において、有限の深さを持つフレームバッファを用いて格子面から展開される画素レベルのソートが行われる。オブジェクト空間のソートは CPU で、画像空間のソートは GPU で実行される。HAVS は 140 万個の四面体格子に対して 1.3fps の計算速度を達成したが、フレームバッファの深さが十分でない場合には描画順序の誤りに基づくアーティファクトが生じる。この深さに対する最適値を決定するのは困難である。格子面のソートでは格子の数のほぼ 2 倍の数のメモリサイズが必要となる。

Anderson ら [28] は、不規則格子向けボリュームレンダリングを実現するためにポイントベース手法を提案した。彼らはひとつの四面体格子をひとつのポイントで表現し、それらをソートし、重ね合わせることによりボリュームレンダリングを実現した。この手法では、140 万、630 万個の格子に対して、それぞれ 5.3fps、0.3fps の計算速度を達成したが、ポイントが画素の各軸に沿った四角形以外とし

て投影される場合には、描画順序の誤りに基づくアーティファクトが生じる可能性がある。

Muigg ら [29] は大規模不規則格子ボリュームデータに対するハイブリッドレイキャスティング手法を提案した。この手法は様々な種類の大規模不規則格子ボリュームデータを Region of Interest (ROI)制御を行いながらレンダリングし、最大で 1500 万個の四面体を 22 秒で処理できた。しかしこの ROI は非興味領域を構造格子に変換してしまう手法であり、加えて曲面から構成される格子を扱うことができなかった。更に、Sakamoto らが指摘している通り [30] 、レイキャスティング法はとりわけ大規模なボリュームデータに対しても有効であるが、ピクセルより小さいサイズの重要な四面体からの寄与を無視する可能性がある。

Roettger ら [25] はグラフィックスハードウェアの性能向上が進むとともに、格子のソート処理が性能上のボトルネックとなると述べた。彼らは、その時点で、グラフィックスハードウェアや特別なハードウェアにおける性能向上によっても、毎秒 150 万個以上の四面体格子を処理できないと予想した。我々はこの予測に賛意を表明し、粒子発光モデルに回帰することにより不規則格子ボリュームデータを処理する手法を提案した。

現在の不規則格子に対するボリュームレンダリング技術は GPU アーキテクチャを用いて高速化を行う場合が多い。Joachim らは GPU を用いた不規則四面体格子向けの包括的なレンダリングパイプラインを提案した [31] 。この技術は有限要素法による構造解析等で計算される、時系列の座標の変位の値を格子頂点に持つボリュームデータの変形する様子をボリュームレンダリングすることができる。Fabio らは不規則四面体格子上に定義された、時系列のスカラー場に対する適応的な可視化の枠組みを提案した [32] 。このシステムは、前処理として、ボリュームデータの圧縮を行い、GPU を用いて現在時刻のボリュームレンダリングを、CPU を用いて次の時刻で必要とされる処理を行い、滑らかな時系列可視化を実現した。CPU では、圧縮されたボリュームデータの解凍、オブジェクト座標上でのソート計算、そして LOD 制御がマルチスレッドで処理される。この手法は、ボリュームデータの圧縮により、もとのボリュームデータと異なった画像が生成されてしまうことが確認された。この手法は、ボリュームデータ全体を GPU メモリ上に格納する必要がある。また、領域分割されたボリュームデータに適用することができない。メモリ容量を超える大きさを持った、領域分割された不規則格子をボリュームレンダリングするためには、PBVR が有効な手法である。

Matthias らは EWA（elliptical weighted average）スプラッティング法を [33] 、Wei らはその手法を GPU 上で高速化するアーキテクチャを提案した [34] 。EWA スプラッティング法は、楕円型ガウシアンフィルタによる再構成核とローパスフィルタをスプラッティング法向けに拡張した技術である。この手法はスプラッティング法に見られた、エイリアシングによる画像のアーティファクトと、過度のぼやけを減少させ、画質を向上させた。この手法はスプラッティングに用いる幾何形状とボリュームデータの両方を GPU メモリ上に格納する必要がある。GPU メモリの容量は年々増加してきているが、数値シミュレーション技術や計算容量の発達に伴い、ボリュームデータもまた巨大化している。また、筆者らはこの手法について、不規則格子に適用可能なスプラッティング手法としているが、論文中で不規則格子に対する具体的な適用例が示されておらず、GPU を用いて高速化を図った続く論文 [34] でも、不規則格子への適用を今後の課題としたままである。

Zhou ら [35] は、大規模不規則四面体格子に適用可能な、ポイントベースのボリュームレンダリング手法を提案した。この手法は高次な補間関数が与えられた場合にも対応した、ソートレスな手法であるが、光の吸収効果を無視した光学モデルを用いている。そのため、正しい奥行き感が表現され得ない。

高い振動数を持つ区分線形な伝達関数が与えられた際に、高画質なボリュームレンダリングを行うために、Ertl ら [15] はテクスチャベースの事前積分法を提案した。先述したとおり、この手法は着目しているサンプリング位置の、手前と奥のスカラー値に対する伝達関数の積分結果を２次元テーブル化(四面体の場合は奥行きも含めた３次元)し、テクスチャとして保持しておくこと。それによって、従来と同じサンプリングの数でより高精度な積分結果が得られる手法である。しかしこの手法はスカラー値を離散化しているため、等値面状の領域に対して穴やクラックのようなアーティファクトが生じる。Stephan ら [36] は階層型構造格子データに対して上述のアーティファクトが生じないボリュームレンダリング手法を提案した。この手法は視線上のスカラー場の関数を解析し極点の位置を求め、それらの極点間で近似と積分を行うことによって、スカラー値を連続的に扱っている。Knoll ら [37] は急峻に変化する部分に加えて、ボリュームレンダリングによる等値面領域を統一的に扱い、高画質にレンダリングする方法を提案した。この手法は区分線形伝達関数の極大点を視線上の関数に対して効率よく探索し、被積分関数のナイキスト周波数を考慮したサンプリングを行い、上述のアーティフ

ァクトの無い高画質なレンダリング結果を得ることができる。しかしながら、この手法を直接不規則格子に対して適用することは困難である。

2.7　粒子ベースボリュームレンダリング

ボリュームレンダリングに関する従来手法が輝度値方程式を何らかの手段で数値計算しているのに対して、その方程式を成立させている密度発光モデルに立ち返ることで得られた手法が粒子ベースボリュームレンダリングである。これまでに述べたように、従来のボリュームレンダリングは、画素ベースや格子ベース、粒子ベースに関わらず、半透明な何か（サンプリング点や要素）を考え、それらを視線上奥行き順に重ね合わせるという計算を行う手法である。このため、大規模不規則格子のボリュームレンダリングを行うときにはサンプリング点の奥行きソートのための計算コストがボトルネックになる。従来手法でソート処理が必要とされる原因は、密度発光モデルにおける光の移送を微分方程式で記述したことにある。

この問題を解決するために Sakamoto と Koyamada は PBVR の基本的アイデアを提案した [30] 。それは輝度値方程式を計算する代わりに、密度発光モデルで起こっている現象をそのまま摸倣するという考えである。つまり、PBVR は不透明で微小な自己発光する粒子をボリュームデータ内部に生成し、粒子同士による光の遮蔽を考慮にいれつつ粒子（厳密には粒子の光）を画像面に投影し、レンダリング画像を得る手法である。このような密度発光モデルを構成する粒子を陽に扱うボリュームレンダリングは PBVR 以前には存在しなかった。

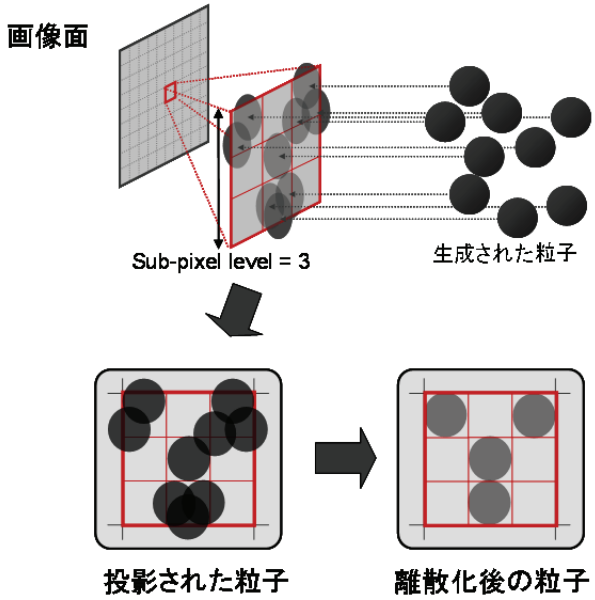


図 2.9　粒子投影と投影像の離散化

粒子ベースボリュームレンダリングは三つの処理ステップから構成される(図 2.9)。第一のステップは、ユーザー指定の伝達関数の不透明度に従って、粒子を発生させることである。第二のステップは、生成された粒子を画像面に投影することである。第三のステップは、投影された粒子の輝度値の平均化により最終画素値を計算することである。

生成された粒子を画素内においてできるだけ正確な場所に投影できるよう、画素をいくつかの仮想的な部分画素（サブピクセル）に分割する。ピクセルの分割数のことをサブピクセルレベルと呼び、図 2.9 に示すように、各ピクセルはサブピクセルレベルの 2 乗個のサブピクセルから構成される。投影計算を行う際、粒子の座標はでサブピクセル位置に離散化される。このサブピクセルは、投影された粒子の輝度値と奥行きの情報を格納するために細分化されたフレームバッファである、粒子バッファによって実現される。粒子バッファの大きさは、フレームバッファ大きさのサブピクセルレベルの 2 乗倍である。このサブピクセルへの粒子の投影と粒子バッファを用いた枠組みのことをサブピクセル処理と呼ぶ。

PBVR はボリューム空間に発生させた粒子からの投影像を、画像面で加算することにより対応する画素における輝度値を計算することができる。PBVR において粒子は完全に不透明なので、視線上の奥にある粒子からの光は手前の粒子により遮蔽される。この効果は Z－バッファアルゴリズムを使って実現することができ、ボリュームレンダリングで一般的に必要とされる投影前のソート処理やアル

ファブレンディングは不要である。

粒子遮蔽の計算はサブピクセル毎に行われ、一つの画素を構成するサブピクセルでの投影像による輝度値を平均化してその画素値を決定する。遮蔽計算により各サブピクセルでは視点より最も近い粒子からの投影像が反映され、すべての粒子投影が終わってから、粒子バッファ上でサブピクセルの平均計算が実行される。

従来の PBVR は、ユーザー指定の伝達関数から計算される不透明度に従って、粒子を生成する。Sakamoto らは、不透明度に比例した粒子密度分布を形成するために、メトロポリス法 [38] を用いた。メトロポリス法は、マルコフ連鎖モンテカルロ法の一種であり、粒子の現在位置と、次に発生させる粒子の候補位置との、エネルギーの割合に比例した確率で粒子を発生させ、任意の粒子分布を実現する手法である。従来の PBVR では、不透明度をエネルギーとみなして粒子の発生を行った。

しかし、不透明度に比例した粒子分布を用いる従来の PBVR は、発生粒子数とピクセルの分割数をユーザーが任意に指定できるため、伝達関数に対応する画像を一意に決定することが困難であった。発生粒子数を増やすほど画像の透明度は減り、ピクセルの分割数（サブピクセルレベル）を増やすほど画像の透明度が増した。伝達関数から画像を一意に決定するために、粒子数を推定するための手法が必要である。

また、Sakamoto らの提案では、構造格子を対象としていたため、不規則格子に対する粒子発生手法について議論がなされていなかった。従来の PBVR では、ボリュームデータ全体にサンプリング候補点をランダムウォークさせていた。しかし、不規則格子に対してこの方法を適用するためには、格子の隣接情報が必要となり、格子形状に応じたメモリリソースが必要となる。大規模な不規則格子ボリュームデータをレンダリングするためには、格子の隣接情報に依存しない粒子発生法が必要である。

第3章　粒子モデルに基づいた高画質化手法

従来の PBVR は、ユーザー指定の伝達関数から計算される不透明度に従って、粒子を生成する手法である。Sakamoto らは、不透明度に比例した粒子密度分布を形成するために、メトロポリス法 [38] を用いた。メトロポリス法は、マルコフ連鎖モンテカルロ法の一種であり、粒子の現在位置と、次に発生させる粒子の候補位置との、エネルギーの割合に比例した確率で粒子を発生させ、任意の粒子分布を実表する手法である。従来の PBVR では、不透明度をエネルギーとみなして粒子の発生を行った。

しかし、不透明度に比例した粒子分布を用いる従来の PBVR は、発生粒子数とピクセルの分割数をユーザーが任意に指定できるため、画像の不透明度を一意に定めることが困難であった。つまり、図 3.1 に示すように、発生粒子数を増やすほど画像の不透明度が増加し、サブピクセルレベルを増やすほど画像の不透明度が減少した。

図 3.1 において、サブピクセルレベル 3、粒子数 10^5 の画像はデータ内部の赤色の部分が可視化されているのに対し、サブピクセルレベル 2、粒子数 10^6 の画像はデータの形状が明確に確認できる。このように、不透明度が変化するとレンダリング結果の特徴が変化する。伝達関数によって指定されたユーザーの意図を正確に反映するためには、粒子数を推定し、伝達関数から画像を一意に決定するための手法が必要である。

また、Sakamoto らの提案では、構造格子を対象としていたため、不規則格子に対する粒子発生手法について議論がなされていなかった。従来の PBVR では、ボリュームデータ全体にサンプリング候補点をランダムウォークさせていた。しかし、不規則格子に対してこの方法を適用するためには、格子の隣接情報が必要となり、格子形状に応じたメモリリソースが必要となる。大規模な不規則格子ボリュームデータをレンダリングするためには、格子の隣接情報に依存しない粒子発生法が必要である。

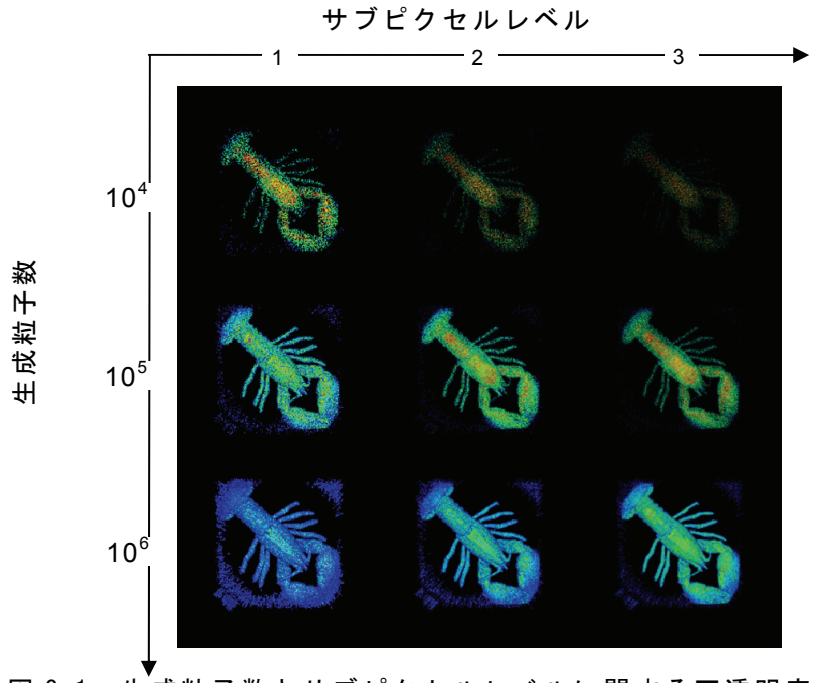


図 3.1　生成粒子数とサブピクセルレベルに関する不透明度の変化

この問題を解決するために、粒子発光モデルから導出した粒子密度推定法を導入し、伝達関数から画像を一意に決定する方法を提案する。そして、粒子密度推定法を用いて、PBVR を四面体格子からなる不規則格子ボリュームデータに適用する方法について述べる。粒子密度推定式に従う PBVR は密度発光モデルに正しく従っているため、十分に小さい粒子を用いて生成された画像はボリュームレンダリングの“正解”の画像に近づくと考えられる。

3.1　粒子密度推定式の導出

伝達関数から一意に画像を作り出せない原因は、不透明度が粒子密度と等価と解釈され、加えて、サブピクセルレベルが粒子発光モデルから意味づけされていないためである。この問題を解決するために、粒子モデリングの導入によりサブピクセルレベルと粒子形状を関係付け、粒子発光モデルから粒子密度推定式を導出する。

我々の粒子モデルでは、粒子について三つの属性、すなわち、形状・大きさ・密度を考慮する。粒子形状は、計算を効率化するために、すべて同一半径をもつ球とする。粒子の大きさは有限とし、球の半径 r で規定され、ピクセルよりも小

さな粒子が分布していると考える。粒子半径を計算するために、ウインドウ座標上で定義されたピクセル幅 1 から、オブジェクト座標におけるピクセル幅 p_{obj} を求める。そして p_{obj} をサブピクセルレベル(L_S)で除したものが粒子の直径となるように定める。粒子半径 r は以下の式で計算される。

$$r = \frac{p_{obj}}{2L_S} \tag{3.1}$$

また、粒子を表表するためにサブピクセルレベルに従って分割された $level^2$ 個の部分領域をサブピクセルと呼ぶ。

粒子密度ρは、粒子半径 r 、ユーザーによって定められた伝達関数から計算される不透明度値α、そしてボリュームレイキャスティングで使用されるレイセグメント長さ Δt を使って計算できる。式(2.23)より以下のように求められる。

$$\rho = \frac{-\log(1-\alpha)}{\pi r^2 \Delta t} \tag{3.2}$$

PBVR によりボリュームレイキャスティング画像と同等の画像を生成するために、粒子密度分布は式(3.2)の関係に従って推定される必要がある。式(3.2)からサブピクセルレベルを 2 倍にするとそれに応じて粒子密度をその 2 乗、すなわち 4 倍にする必要があることが理解できる。

空間点過程においてハードコア過程 [39] を仮定すると、粒子密度 ρ にはその上限 $\rho^{\max}$ が存在する。PBVR において、粒子位置はサブピクセルの位置に離散化される。そのため、最大密度は、粒子を外包する最小立方体で考える。最小立方体の体積は $8r^3$ となるので、次の関係式が成立する。

$$\rho^{\max} = \frac{1}{8r^3} \tag{3.3}$$

従って、不透明度にも対応する上限 $\rho^{\max}$ が存在する。不透明度が $\rho^{\max}$ から 1 までの値をとる場合には対応する密度は一定値 $\rho^{\max}$ となる。ここで式(2.23)から $\rho^{\max}$ は以下のように求められる。

$$\alpha^{\max} = 1 - \exp\left(-\pi r^2 \rho^{\max} \Delta t\right) \tag{3.4}$$

以上の内容をまとめると、粒子密度推定式は以下のようになる。

$$\rho = \begin{cases} \dfrac{-\log(1-\alpha)}{\pi r^2 \Delta t} & (0 \le \alpha \le \alpha_{\max}) \\ \rho_{\max} & (\alpha_{\max} \le \alpha \le 1) \end{cases} \tag{3.5}$$

この粒子密度推定式(3.5)により、伝達関数から計算される不透明度 αから一意に粒子密度が決定される。

3.2　四面体格子に対する粒子生成

導出した粒子密度推定式を用いて、格子を構成する四面体要素毎に粒子数を推定することによって、不規則格子ボリュームデータに対して粒子生成を行うことができる。粒子生成には、ひとつの四面体要素がメモリに格納されているだけでよい。先ず式(3.5) より得られる密度 ρ から発生粒子数を計算し、次に確率統計的な手法で粒子を分布させる。生成された粒子はそのまま次のステップ、粒子投影、サブピクセル処理へと受け渡される。以下の節では、任意の位置の粒子密度を求めるために必要なボリュームデータ内部のスカラー値補間法について述べ、そして粒子数計算法について述べる。最後に粒子の分布方法について述べる。

3.2.1　スカラー値の補間

ボリュームデータのスカラー値は離散的な格子上に定義されるため、節点以外の位置でのスカラー値が不明である。しかしボリュームレンダリングを実行するためには、格子内部の任意の位置におけるスカラー値が必要とされる。本提案手法においても粒子分布を得るために、任意の位置での粒子密度を計算する必要がある。そのために、ボリュームデータ内部の任意の位置 $\mathbf{x} = (x,y,z)$ でのスカラー値 $s(\mathbf{x})$ を用いて、伝達関数から不透明度 $\alpha(s)$ を計算し、粒子密度推定式(3.5) に代入して粒子密度の値を得る。本研究では四面体格子内部のスカラー値は、四面体要素の頂点上のスカラー値とその位置から線形補間を用いて計算される。線形補間を用いるとき、位置 $\mathbf{x}$ でのスカラー値 s は係数 (a,b,c,d) を用いて、1 次関数 $s = ax + by + cz + d$ で計算される。四面体要素が、図 3.1(a) に示されているように、頂点 v_0, v_1, v_2, v_3 から構成されているものとする。ここで頂点の座標を $v_i = (x_i, y_i, z_i)$、そして頂点 v_i 上のスカラー値を s_i とする。ここで i=0,1,2,3 である。1 次関数の係数(a,b,c,d) は以下の式で計算される。

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} x_0 & y_0 & z_0 & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} \tag{3.6}$$

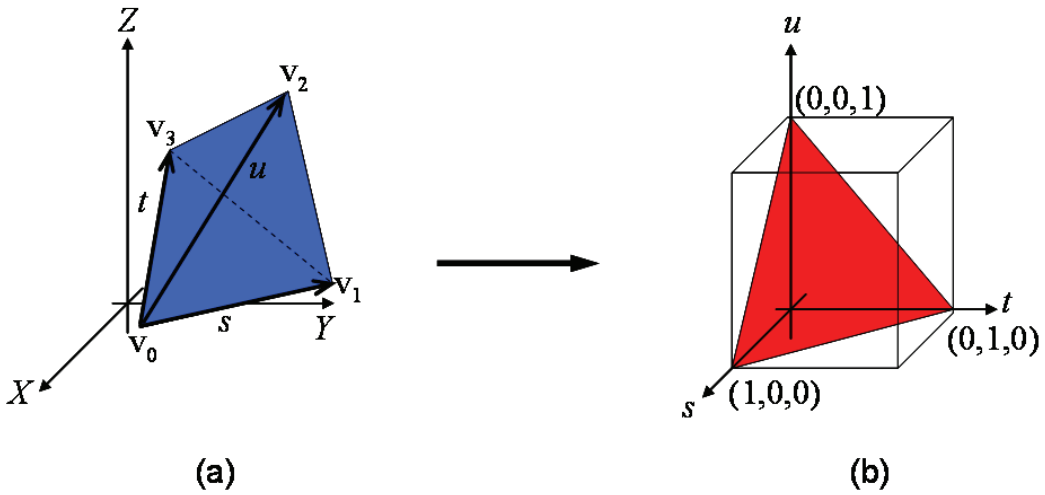


図 3.1　四面体要素と計算空間

3.2.2　一様分布の形成

生成した乱数を利用して任意の形状の四面体要素内部に粒子を一様分布させる方法を述べる。一様乱数を発生させる手法は多く知られているが [40] [41] ,生成された乱数の間に相関が生じないような手法であれば何を用いても良い。[0, 1] の区間に発生する三つの独立な一様乱数 r_0, r_1, r_2 を組み合わせることによって領域 $[0, 1]^3$ の内部に一様分布する乱数を作り、これを四面体内部にマッピングして四面体セル内部の一様分布を得る。位置ベクトル V を、四面体要素の節点 v_i とパラメータ s, t, u を用いて以下のように表す。

$$V = (v_1 - v_0)s + (v_3 - v_0)t + (v_2 - v_0)u \tag{3.7}$$

このとき s, t, u の値が以下の両方の条件を満たせば V は四面体内部に収まる(図 3.1(b) 参照)。

$$0 \le s,t,u \le 1 \tag{3.8}$$

$$s + t + u \le 1 \tag{3.9}$$

s, t, u に一様乱数 r_0, r_1, r_2 を対応させれば、乱数が式(3.11) を満たすとき四面体内部に一様乱数が得られる。

この方法は乱数が式 (3.9)の範囲から外れた場合に四面体内部に V が収まらず、乱数を生成しなおす必要がある。V が範囲内に入る割合は、図 3.1 における立方体内部の四面体の体積と一致し、1/6 しかない。

この問題を解決するために、*s, t, u* の値が式(3.9) を満たさないとき、アフィン変換を用いて領域外の乱数を四面体内部に写像する。乱数 r_0, r_1, r_2 が分布する立方体状の領域を図 3.2 に示すように五つの四面体に分割する。そして、図 3.1(b) 以外の四面体領域を, アフィン変換により回転, 変形させて, 図 3.1(b) の四面体領域に一致させる。変換式を以下に示す。

$$\begin{bmatrix} s \\ t \\ u \end{bmatrix} = \begin{bmatrix} r_0 \\ r_1 \\ r_2 \end{bmatrix} \quad r_0 + r_1 + r_2 \le 1 \tag{3.10}$$

$$\begin{bmatrix} s \\ t \\ u \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -1 & 1 \\ -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ 1 \end{bmatrix} \quad r_0 - r_1 + r_2 \ge 1 \tag{3.11}$$

$$\begin{bmatrix} s \\ t \\ u \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ 1 \end{bmatrix} \quad r_0 + r_1 - r_2 \ge 1 \tag{3.12}$$

$$\begin{bmatrix} s \\ t \\ u \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ 1 \end{bmatrix} \quad -r_0 + r_1 + r_2 \ge 1 \tag{3.13}$$

$$\begin{bmatrix} s \\ t \\ u \\ 1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & 1 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ 1 \end{bmatrix} \quad \text{else} \tag{3.14}$$

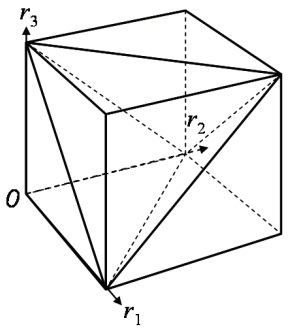


図 3.2　計算空間の四面体分割

3.2.3 生成粒子数の計算

大規模不規則格子ボリュームデータへの適用を想定しているため、画素に対し

て四面体格子が十分に小さいと考えられる。そのため四面体格子内部で一様な密度場を仮定した。この密度の値は、着目している四面体格子の重心位置における密度値 ρ_g で代表させた。

このため、粒子は四面体格子の内部に一様分布するように生成され、発生粒子数 N は以下の式(14)ように、四面体の体積 V_{Cell} と密度の代表値 ρ_g との積で計算される。

$$N = \int \rho \, dV = \rho_g V_{Cell} \tag{3.15}$$

式(14)における粒子数が整数でないとき、最終的にその格子で発生する粒子数 n は以下のように決定される。

$$n = \begin{cases} \lfloor N \rfloor + 1 & \text{if } R \le N - \lfloor N \rfloor \\ \lfloor N \rfloor & \text{otherwise} \end{cases} \tag{3.16}$$

ここで、 R は $[0,1)$ における一様乱数である。

粒子生成の擬似コードを以下に示す。

Algorithm 1: ParticleGeneration ()

```
1:  for each tetrahedral cell T_i do
2:    n = T_i.getNumOfParticles();
3:    s = 0;
4:    for each j = 0,1,...,n do
5:      for each k = 0,1,2,3 do
6:        b_k = random();
7:        s += b_k;
8:      end for
9:      for each k = 0,1,2,3 do
10:       b_k /= s;
11:     end for
12:     Generated particle P_j
13:     P_j.x = Σb_k x_k;   P_j.n_x = Σb_k n_xk;
14:     P_j.y = Σb_k y_k;   P_j.n_y = Σb_k n_yk;
15:     P_j.z = Σb_k z_k;   P_j.n_z = Σb_k n_zk;
16:   end for
17: end for
```

本手法では、四面体格子内部に発生させた粒子を即座にパーティクルバッファに投影し、処理の終わった四面体の情報を破棄する。このように、提案手法では、着目四面体格子をストリーミング的にメモリに渡すことによって、大規模不規則格子ボリュームデータを構成する全格子情報を保持しておく必要がない。したが

って、利用可能なメモリ容量では格納不可能な大規模なデータであっても、ボリュームレンダリング処理が可能となる。

3.3 画質に関する検証

粒子数の計算に粒子密度推定式を導入することで、サブピクセルレベルをパラメータとした画像生成が可能になった。この手法によって伝達関数に対する一意の画像生成が可能になったか否かを検証するため、従来手法で生成された画像との比較を行った。そして、サブピクセルレベルが画質に与える影響を調査するため、ゆらぎ（画像の標準偏差）の値を指標とした解析と実験を行った。また、PBVR は粒子投影後に粒子位置をサブピクセル位置に対して離散化するが、この手法が画質に対して与える影響について、簡単なモデルを用いた解析を行った。本手法は KVS（Kyoto Visualization System）ライブラリを用いて実装された。KVS はマルチプラットホーム対応のオープンソフトウェアで構成された科学技術と可視化計算向き C++ライブラリである。実験では、3.2GHz Intel Pentium XE と 2GB の RAM を搭載した PC を使用した。計算された画像解像度は 512^2 である。

3.3.1 レイキャスティング画像との比較

粒子密度推定式を導入して生成された画像の品質を評価するために、同一の伝達関数を使って計算されたレイキャスティングの画像と比較した。レイキャスティングは高品位な画像を生成することで知られる手法である。提案手法の粒子直径を表すサブピクセルレベルを 1 から 7 まで変化させて画像を生成した。そして定量的な評価を行うため、レイキャスティング画像に対する誤差の値を計算した。

図 3.3 は各サブピクセルレベルに対する画像を並べたものであり、右端の画像はレイキャスティングによる画像である。また、表 3.1 にサブピクセルレベルに対する生成粒子数、粒子生成時間、レンダリング速度を示す。図 3.3 から、サブピクセルレベルを増加させることで、PBVR による画像が滑らかになり、レイキャスティングによる画像に近付いていることが確認できる。また、異なるサブピクセルレベルの画像の間に画質の差が認められるが、同様の不透明度を持つ画像を生成している。つまり、サブピクセルレベルを通して粒子半径 r を変更するこ

とによって、ボリュームレイキャスティングの画像に類似した画像のまま、画像詳細度（Level of detail, LOD）だけが変化していることが確認できる。

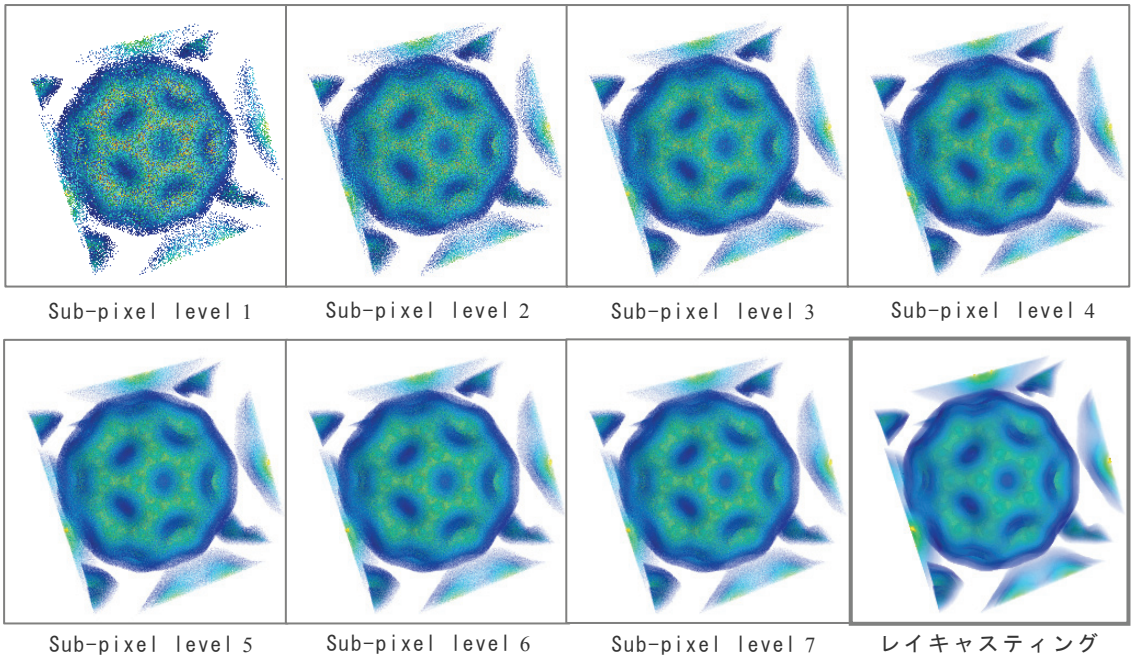


図 3.3　サブピクセルレベルに対する Bucky(要素数 1.25M)のレンダリング画像

表 3.1　サブピクセルレベルに対する
生成粒子数、粒子生成時間、レンダリング速度

Sub-pixel level	1	2	3	4	5	6	7
Nparticles [M]	0.2	1.0	2.4	4.8	8.2	12.4	17.7
Sampling time [msec]	2.1	6.0	8.3	10.0	13.0	15.8	21.1
FPS	23.6	6.5	2.9	1.5	0.9	0.6	0.4

PBVR により生成された画像の品質を評価するために、ボリュームレイキャスティングで生成された画像と PBVR で生成された画像との色空間における距離を計算し誤差の値とした。誤差 E_{image} は以下のように定義される。

$$E_{image} = \sum_{i}^{L} \frac{e_i}{L} \tag{3.17}$$

$$e_i = \sqrt{\Delta R_i^2 + \Delta B_i^2 + \Delta G_i^2} \tag{3.18}$$

ここで、L は画素数、 $\Delta R, \Delta G, \Delta B$ は画像における赤、緑、青成分における差を表す。図 3.4 にサブピクセルレベルに対する誤差と計算時間との関係を示す。この図より、サブピクセルレベルを上げるにしたがって誤差が小さくなることが確

認できる。

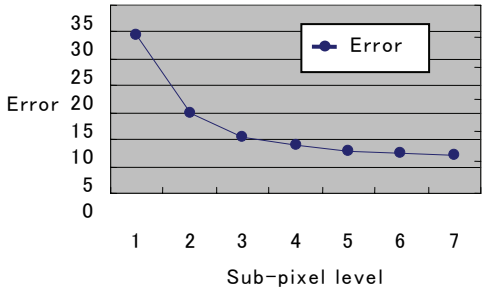


図 3.4　サブピクセルレベルに対する誤差

3.3.2 粒子位置の離散化による誤差の検証

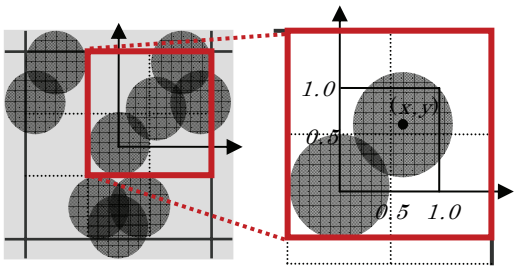
粒子は画像面上に投影され、投影された粒子の座標は計算を高速化するためにサブピクセル位置に離散化される。一方、ボリュームレイキャスティングでは、レイは視点とピクセル中心を通る直線として定義されるのでこのような離散化処理は不要となる。この離散化処理はアーティファクトを引き起こす可能性がある。投影された粒子がそれぞれ直径以上離れている場合は、アーティファクトはあまり目立たないが、離散化前に粒子像に重複領域が存在する場合は、離散化処理の影響により投影面積の和に違いが生じる可能性がある。

この投影面積の違いによる影響を評価するためにふたつの粒子の投影像が単一画素中に含まれる場合を想定し、投影像の面積の総和を計算する（図 3.5）。ふたつの投影像に重なりがない場合、その総和はひとつの粒子の投影像の二倍である。また、完全に重複している場合にはひとつの粒子の投影像の面積に一致する。投影像の形状を半径 0.5 の円とし、そのうち一方の中心が原点に固定されていると仮定する。もう一方の円の中心座標を (x, y) とする。対称性より、座標値 (x, y) は領域 $[0,1]^2$ に含まれるとする。我々は投影像同士の重なりの影響を考察したいので、 (x, y) は領域内で一様な乱数であると仮定する。離散化処理を行わない場合の投影像の面積の総和の期待値 S_{wo} と、離散化処理を行なう場合の投影像の面積の総和の期待値 S_w について、それぞれ以下のように計算された。

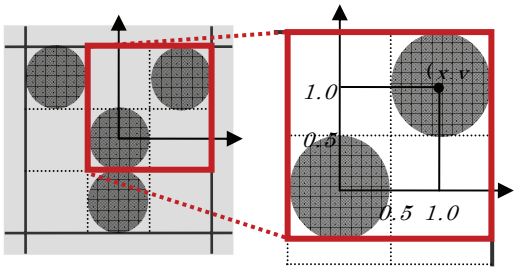
$$
\begin{aligned}
S_{wo} &= \int_0^1\int_0^1 \frac{\pi}{2} - \frac{1}{2} \times \max\left(\arccos(q) - q\sqrt{1-q^2}, 0\right) dx\,dy \\
&= \frac{\pi}{2} - \frac{\pi^2}{64} \\
S_w &= \frac{\pi}{2} \times \frac{3}{4} + \frac{\pi}{4} \times \frac{1}{4} \\
&= \frac{7\pi}{16}
\end{aligned}
\tag{3.19}
$$

式(3.19)において、q はふたつの投影像間の距離を示す。

S_w / S_{wo} = 0.97 となったので、離散化処理の有無は投影像の面積の総和にそれほど大きな影響を及ぼさないと考える。



(a) 粒子投影像



(b) 粒子投影位置の離散化

図 3.5　粒子投影像とサブピクセル位置に関する離散化

3.3.3　画像のゆらぎの検証

PBVR は粒子位置の不確かさによるゆらぎの影響を受ける。粒子密度分布は統計的手法で実現されるため、その位置は一意に決定されない。そのため、異なる乱数を用いてサンプリングした粒子群から複数の画像を生成し、それらを並べてすばやく切り替えることで画像にちらつきが観察される。本論文ではこの画像のちらつきのことをゆらぎと呼ぶ。ボリュームレンダリングにおいて、レンダリングの度に画像が異なるような、画像間のゆらぎは無いことが望ましい。本提案手法において、粒子を小さくする、すなわちサブピクセルレベルを上げることによ

ってゆらぎの影響が抑えられることが期待される。

サブピクセルレベルの増加によるゆらぎの低減を定量的に示すために、ピクセルの輝度値に関する標準偏差を用いる。ゆらぎの原因は、同一の位置にあるピクセルの輝度値が画像間で異なることを意味する。そしてそれらの値が大きく変動するほどゆらぎが大きい。このような値の振れ幅を計測するためには、ピクセルの輝度値に関する標準偏差を用いるのが妥当である。

ゆらぎについて検証するため、まず簡単なモデルを用いて標準偏差について解析し、サブピクセルレベルと標準偏差の関係式を導出する。そして実データに対して提案手法を適用して標準偏差を計測し、導出した関係式の妥当性を示す。サブピクセルレベルとゆらぎの関係を明確にすることで、ユーザーが任意の画質でレンダリングを行うための指標になりうる。

3.3.3.1　サブピクセルレベルとゆらぎの関係

幅が 1 で長さ Δt のレイセグメント上での輝度値を計算する。式(3.5)より生成する粒子数は r^2 に、すなわち $k \cdot L_S{}^2$ に逆比例する。ここで k は式(3.4)を式(3.5)に代入して得られる変数を L_S とした場合の定数項である。レイセグメントにおける粒子密度は次のように表せる。

$$\rho = \frac{k \cdot L_S{}^2}{1 \cdot 1 \cdot \Delta t} = \frac{-\log(1-\alpha)}{\pi\left(\frac{1}{2 \cdot L_S}\right)^2 \Delta t} \tag{3.20}$$

従って、不透明度は次のように表現される。

$$\alpha = 1 - \exp\left(-k\pi/4\right) \tag{3.21}$$

粒子半径の縮小によるゆらぎの低減効果を理論的に示す。輝度値は画像面上における粒子群の投影面積としても計算することができる。第 2 章で述べたように、不透明度 α は粒子発光が遮蔽される確率、すなわち一つ以上の粒子がレイセグメント内部に存在する確率を表す。よってサブピクセルの輝度値を確率変数（粒子が投影されたときの値を 1、されなかったときの値を 0）として平均値は以下のように計算される。

$$B_{Ave} = 1 \cdot \alpha + 0 \cdot (1-\alpha) = \alpha \tag{3.22}$$

同様に分散も以下のように計算される。

$$B_{Var} = (1-\alpha)\alpha \tag{3.23}$$

最終的なピクセルの輝度値は全てのサブピクセルの値を平均することによっ

て得られる。i 番目のサブピクセルの輝度値を B^i とし、各サブピクセルの値は互いに独立であるとする。ピクセルの値はサブピクセルの値から次のように計算される。

$$B^{total} = \sum_{i=1}^{level^2} B^i / {L_S}^2 \tag{3.24}$$

ピクセルの値の分散は、輝度値 B^i を確率変数として、以下のように展開される。

$$\begin{aligned} & Var\left(B^{total}\right) \\ &= \frac{1}{{L_S}^4}\left\{\sum_{i=1}^{{L_S}^2} Var(B^i) + 2\sum_{i,j\ i<j} Cov(B^i, B^j)\right\} \\ &= \frac{1}{{L_S}^4}\sum_{i=1}^{{L_S}^2} Var\left(B^i\right) \end{aligned} \tag{3.25}$$

粒子発光モデルでは、粒子分布についてポアソン分布が仮定されているため、投影される粒子は互いに影響を及ぼしあわない。そのため輝度値 B^i は互いに独立になり、共分散 $Cov(B^i,B^j)$ の値は 0 になる。また B^i の分散は式(3.23)より決定される。よってピクセルの値の標準偏差は以下のようになる。

$$B_{Dev}^{total} = \frac{\sqrt{(1-\alpha)\alpha}}{L_S} \tag{3.26}$$

上式より、ピクセルの分散の値がサブピクセルレベルに反比例して減少していくことが言える。

3.3.3.2　実データを用いた検証

サブピクセル処理によるゆらぎの抑制効果を検証するために、提案手法による画像のゆらぎを計測した。粒子発生に用いる乱数のシードを変えた画像を確率場 $\mathbf{I}(i,j)=(R(i,j),G(i,j),B(i,j))$ の実現値とみなし、分散 $Var(\mathbf{I}(i,j))$ を計算した。（R、G、B はそれぞれ 0~255 の赤、緑、青の値、i 、j は画像座標を表す。）そして標準偏差の代表値 $\sigma^{\mathbf{I}}$ を以下の式で定義した。

$$\sigma^{\mathbf{I}} = \sum_{i,j} \frac{\sqrt{{\sigma_{R(i,j)}}^2 + {\sigma_{G(i,j)}}^2 + {\sigma_{B(i,j)}}^2}}{L} \tag{3.27}$$

ここで L はボリュームが存在する領域の画素数、$\sigma_{R(i,j)}$ 、$\sigma_{G(i,j)}$ 、$\sigma_{B(i,j)}$ はそれぞれ $R(i,j)$ 、$G(i,j)$ 、$B(i,j)$ の標準偏差である。表 3.2 に示すボリュームデータから、Spx、Fighter、Blunt、Aorta を用いて実験を行い、各サブピクセルレベルに対する標準偏差の代表値$\sigma^{\mathbf{I}}$を計算した。実験には乱数のシードを変えた 20 枚の標本を用いた。それぞれのデータに関して、図 3.6 に標準偏差の代表値をプロットしたグラフ、図 3.7 にサブピクセルレベル 7 のレンダリング結果を示す。こ

のグラフから、全てのデータについてサブピクセルレベルの増加に対する標準偏差の減少が確認できる。

表 3.2　ボリュームデータの四面体数と節点数

Dataset	Num. of tets	Num. of nodes
Spx	12,936	2,896
Fighter	70,125	13,832
Blunt	222,414	40,948
Post	616,050	108,300
Aorta	1,386,882	248,992
Sf2	2,067,739	378,747

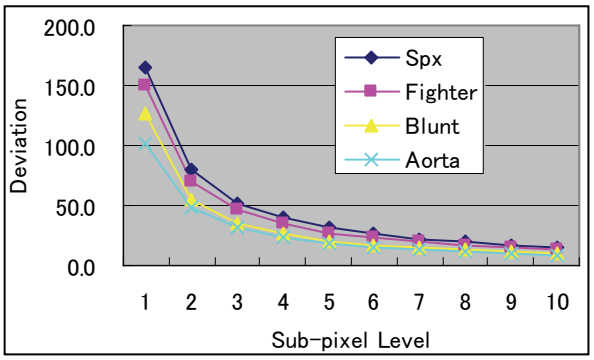
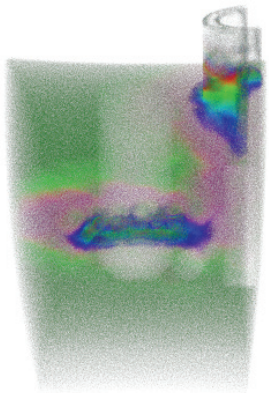
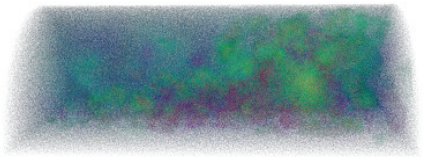


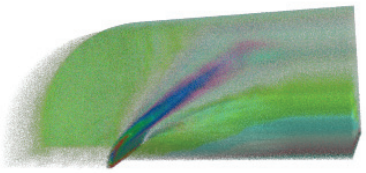
図 3.6　サブピクセルレベルに対する標準偏差



(a) Spx



(b) Fighter



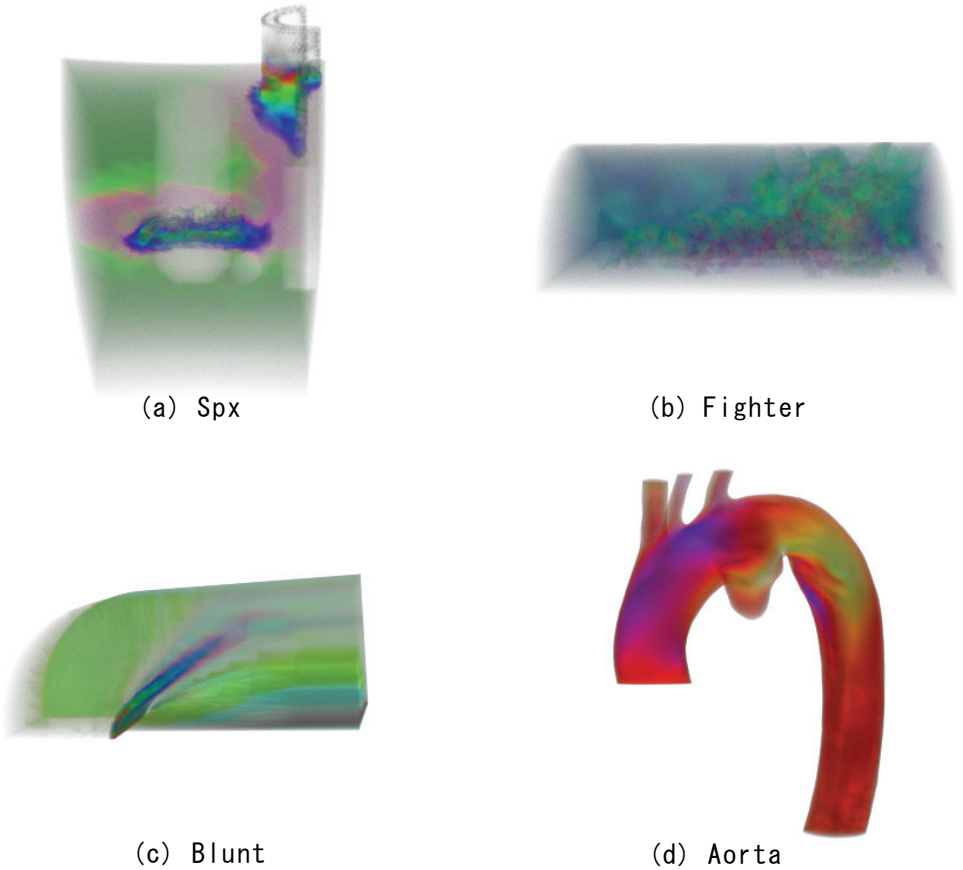
(c) Blunt

(d) Aorta

図 3.7　サブピクセルレベル 7 によるレンダリング画像

また、分散を計算する課程で生成された、平均の画像について、画質が改善されるという現象が確認された。図 3.8 にはサブピクセルレベル 7 で生成された画像を 20 枚平均した結果の画像を示す。図 3.7 と比較すると、不透明度を変化させ

ること無く、明らかに画質が改善されていることが確認できる。



(a) Spx　(b) Fighter

(c) Blunt　(d) Aorta

図 3.8　サブピクセルレベル 7 の画像 20 枚を用いた平均画像

3.4　適用例

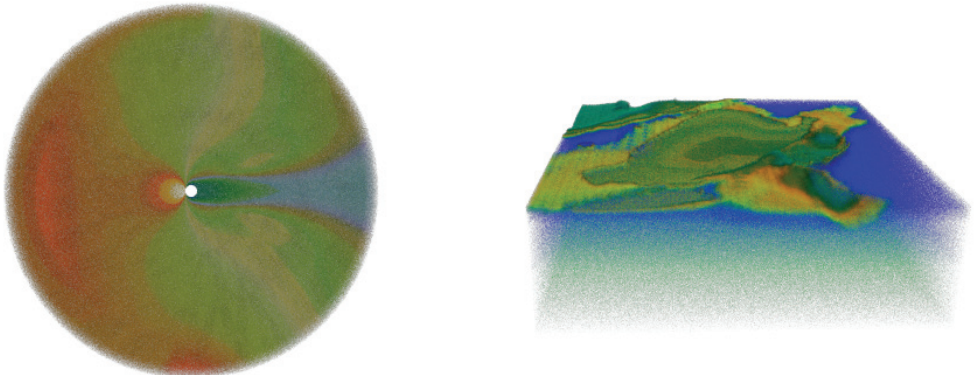
3.4.1　計算コスト

本手法の計算コストを確認する実験を行った。この実験では、生成した粒子をメモリ上に保持してレンダリングを行った。そして、生成粒子数と粒子生成時間、レンダリング時間を計測した。表 3.2 に掲載する全てのボリュームデータに対して実験を行った。また、ボリュームレイキャスティング画像に対して画質を落とさないように、サブピクセルレベルを 7 に設定した。表 3.3 に計測結果を示す。Post と Sf2 のレンダリング画像を図 3.9 に示す。粒子生成時間とレンダリング時間が生成粒子数に比例して増加していることが確認できる。生成粒子数の増加に

より粒子投影処理が行われる回数も増加するためレンダリング時間が増加していると考えられる。生成粒子数については伝達関数から導出される粒子密度関数と、サブピクセルレベルに依存すると考えられる。

表 3.3　生成粒子数と粒子生成時間、レンダリング時間

Dataset	Num. of particles	sampling time [msec]	rendering time [msec]
Spx	1.8M	759	524
Fighter	2.4M	1018	664
Blunt	3.3M	1572	681
Post	6.7M	3345	1100
Aorta	4.0M	3317	667
Sf2	9.0M	6202	1106



(a) Post　　(b) Sf2

図 3.9　サブピクセルレベル 7 のレンダリング画像

3.4.2 スケーラビリティ

本手法によるスケーラビリティを示すために、表 3.2 に掲載した不規則格子ボリュームデータをもとにして、人工的に大規模な不規則格子ボリュームデータを作成する。ひとつの四面体の各稜線を二等分することにより、合計八つの小四面体を生成する方法を用いて再帰分割を繰り返し、表 3.4 で確認できるように、最大 10 億個の四面体格子からなる不規則格子ボリュームデータを生成した。生成粒子群をメモリ上に保存せずに、四面体要素を読み込むたびに粒子生成と投影を行った。図 3.10 に表 3.4 の処理時間を四面体格子数に対してプロットした結果を示す。図 3.10 より、処理時間は四面体格子数に比例することがわかる。表 3.4 において、データ名のカッコ内の数字は、再分割処理の適用回数を示している。10 億個の四面体格子かなる不規則格子ボリュームデータは、今回実験で使用した PC

のメモリには格納できないことに留意されたい。

表 3.4　分割による四面体数、生成粒子数、粒子生成・投影時間

Dataset	Num. of tets	Num. of particles	processing time [sec]
Spx (2)	0.8M	1.8M	0.96
Post (1)	4.9M	6.7M	3.35
Aorta (1)	11.1M	4.0M	5.76
Fighter (3)	35.9M	2.4M	16.08
Blunt (3)	113.9M	3.3M	48.76
Sf2 (3)	1.0G	9.0M	441.39

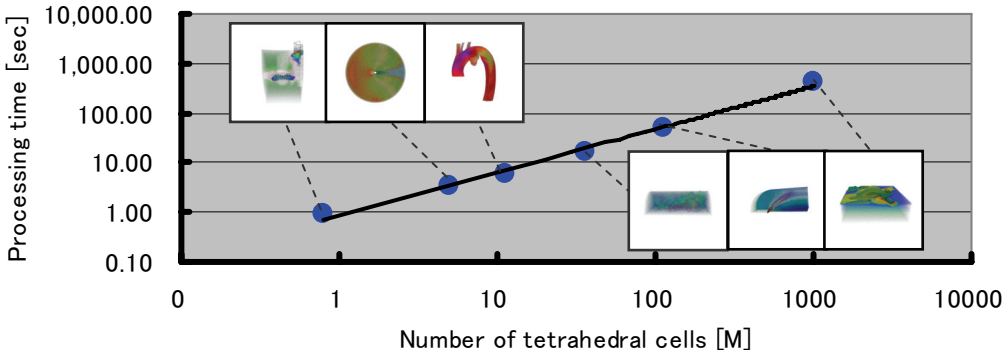


図 3.10　レンダリング画像と四面体数、粒子生成・投影時間

3.5　考察

提案手法により、メモリに格納できないサイズの大規模不規則格子ボリュームデータをプレビュー用としてレンダリングできることが確認できた。従来的なボリュームレイキャスティングや四面体投影法は、計算時に全ての格子データをメモリに格納したうえで、隣接格子情報や視線に沿った格子のソーティングの計算が必要とされるため、本実験と同一環境でのレンダリングは困難である。また、提案手法は理論上レンダリングできるデータサイズに上限が無いため、実験に用いた 10 億個を超える数の四面体からなるボリュームデータのレンダリングも可能だと考えられる。

提案手法の計算コストは粒子数に影響を受ける。この粒子数は可視化パラメータが変化した場合に変化する。伝達関数や世界座標系におけるピクセルサイズが変化した場合、密度関数が変更され、生成すべき粒子数が利用可能なメモリで格納が困難となってしまう可能性がある。このような場合、前節で述べたストリー

ミング技術の適用を考えることができる。すなわち、全ての粒子をメモリに格納するのではなく、各四面体格子で生成した粒子のみを一時的にメモリに格納し、画像平面に投影した後廃棄する。この処理を全ての四面体格子に対して行うことで、最終的なレンダリング画像を得ることができる。この手法の延長線上でスプラットを不透明発光粒子で構成するような新しいソート不要のスプラッティング法も開発が可能である。

今回の提案手法では四面体格子における密度は一定とし、四面体の重心の値で代表させた。密度関数が四面体内で大きく変化し、特にその形状のアスペクト比が大きい場合には格子面付近でのアーティファクトが目立つ可能性がある。没入表示空間でよく見られるように画像サイズに比べて四面体が大きく投影される場合が当てはまる。この問題を解決するために格子内における密度分布を考慮する必要が出てくる。これを実表するためには四面体格子内においてメトロポリスサンプリング等を使ってもよいが、計算コストが大きくなる可能性がある。将来的には、計算コストの観点で有効なサンプリング手法の開発を行う予定である。

ソート不要のボリュームレンダリング手法として、例えば、HAVS において、格子面の重心ソートを行わず、バッファのサイズ k を十分に大きくすることも考えられる。しかしながら、最適な k のサイズを決定することが困難である。１億個程度の四面体格子データの処理には、1k レベルの深さが k-buffer に必要とされる。我々の手法はサブピクセルでの輝度値を平均化するだけでよいが、HAVS では画素ごとに k-buffer に格納されたフラグメント値をソートし、アルファ合成する必要がある。

3.6　まとめ

伝達関数に対して画質が一意に定まらないという問題点を解決するために、密度発光モデルから粒子密度推定式を導出し、生成粒子数を計算する手法を提案した。この手法を用いて、ボリュームレンダリングでは高画質な手法として知られるレイキャスティング法と比較可能な画像を生成することができた。サブピクセルレベルを増加させるほど、レイキャスティングの画像に近づくことがわかった。

また、この手法は確率統計的な手法で粒子を生成しているため、画像にゆらぎが生じる。このゆらぎについて検討するため、結果画像の画素値の分散を計算し

た。まず、画素に対する粒子投影に関する簡単なモデルに対して検討を行った。その結果、サブピクセルレベルを上げることによって、分散の値を減少させられることがわかった。そして実データに適用して得られた画像に対して分散の値を算出した。実データに対しても、サブピクセルレベルの増加に伴う分散の減少を確認できた。そして、画像の分散の値を計算する仮定で作成された平均の画像について、画質の改善が発生した。

平均画像の画質について、簡単なモデルを作成し解析を行った。その結果、サブピクセルレベルの 2 乗回だけ平均した画像は、そのサブピクセルレベルと同等の画素値が得られることが判明した。実データに適用し、そのことを確認した。

この手法は従来手法に置き換わるものではなく、従来手法ではレンダリングが困難であったメモリに乗り切らないサイズのデータを、プレビュー用にボリュームレンダリングすることができる手法である。この手法を四面体格子からなるボリュームデータに適用し、10 億格子からなる四面体格子のボリュームレンダリングに成功した。

本手法は利用可能なメモリ容量に殆ど制約を受けず、可視化できる不規則格子ボリュームデータのサイズの大規模化にうまく対応できる。本手法は四面体ごとに任意の順序で粒子の生成と投影を行うため、少ない計算機リソースで大規模データを処理することができる。大規模不規則格子ボリュームデータを使ったシミュレーションについては大規模分散環境で計算されることが多い。そしてシミュレーション結果についても分散環境で出力されており、これらを可視化するために出力データを一ヶ所に集約することは困難である。ストリーミング技術はこのような場合の可視化に重要であり、本提案手法はこれらの要求に対して自然に対応が可能である。

第4章　不規則格子向け粒子ベースボリュームレンダリング

粒子密度推定式と四面体格子に対する一様分布生成手法の提案により要素毎の粒子生成が可能になり、メモリを圧迫する隣接情報が不要になった。その結果、大規模な四面体格子ボリュームデータに対する粒子生成が可能になった。しかし数値シミュレーションの現場では、その用途に合わせて様々な種類の要素が用いられ、四面体以外の格子を持つボリュームデータの可視化の要請も強い。

この章では、いかなる要素から構成される不規則格子ボリュームデータにたいしても適用可能な PBVR の粒子生成手法と、サブピクセル処理よりもメモリ効率の良い画像生成手法を提案する。そして領域分割された大規模不規則格子ボリュームデータに適用し、その有効性を検証する。

4.1　粒子生成手法

粒子生成を行うために、初めに生成粒子数の推定を行い、次に要素内部に粒子分布を形成する。生成粒子数の推定は四面体格子の場合と同様に、要素の体積とその重心位置の密度の値の積で計算する。粒子分布の生成は、計算空間である局所座標で粒子を生成し、オブジェクト座標である大域座標に写像することで実表する。この手法は局所座標から大域座標への写像に、有限要素法で用いられるアイソパラメトリック要素に対する補間関数と形状関数を利用する。そして、格子内部での粒子密度分布を生成するために、局所座標内部でメトロポリスサンプリングを行う。

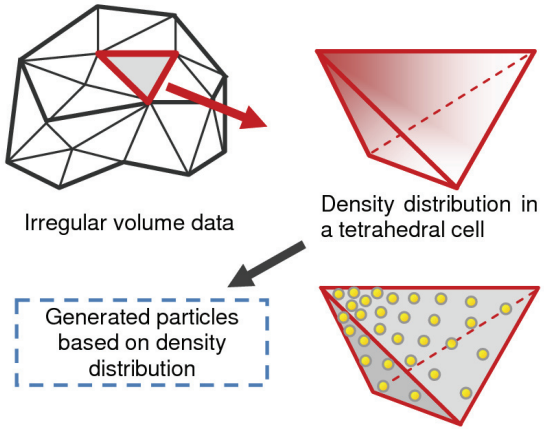


図 4.1　密度分布に比例した粒子群

4.1.1　メトロポリスサンプリング

この手法では、粒子群は不規則格子を構成する各要素において生成される（図4.1）。粒子密度推定式(3.5)に従って粒子を生成するために、メトロポリスサンプリングを利用する。メトロポリスサンプリングは空間中に粒子を一つずつ生成し、任意の確率密度分布に従う密度分布を生成する手法である。初めに、任意の位置に開始点 x_0 を与える。そして、i 番目の粒子の位置 x_i を逐次生成していく。メトロポリスサンプリングによる粒子生成の手順を以下に示す。

1. 現在の粒子位置 x_i から $\rho(x_i)$ を計算する。新しい粒子生成位置の候補 x'を生成する。これは計算空間からランダムに選ばれる。そして $\rho(x')$ を計算する。
2. $\rho(x')$ と $\rho(x_i)$ の比 W を計算する。$W(x_i \to x') \equiv \rho(x')/\rho(x_i)$.
3. もし$W(x_i \to x') \geq 1$ならば x' を新しい粒子位置として採用し、x_i を x_{i+1} に更新する。そして手順 1 に戻る。そうでなければ手順 4 に進む。
4. 一様乱数 $R \in [0,1)$ を生成し、以下の式に従って新しい粒子位置 x_{i+1} を計算する。

$$x_{i+1} = \begin{cases} x' & \text{if } R \leq W(x_i \to x') \\ x_i & \text{otherwise} \end{cases} . \qquad (4.1)$$

そして手順 1 に戻る。

上述の手順は、指定された粒子数に到達するまで繰り返される。また、手順 4

が呼び出され $x_{i+1} = x_i$ となる場合は、粒子数を増加させない。なぜならば、同一の位置に生成された粒子は、レンダリングプリミティブとして用をなさないからである。

4.1.2 補間関数と形状関数

様々な形状の格子内部に対する粒子生成を可能にするために、有限要素法で用いられるアイソパラメトリック要素に対する補間関数と座標変換を利用する。アイソパラメトリック要素とは座標変換に用いる形状関数が補間関数と同様の多項式を利用する要素である。

この手法は初めに局所座標 $\mathbf{x}$ に対して粒子生成を行い、その粒子位置を大域座標 $\mathbf{X}$ で記述される格子 T の内部に座標変換する手法である。例えば、六面体格子の局所座標は立方体の領域 $[0,\ 1]^3$ で、四面体や四面体二次要素の場合は体積座標で記述され、局所座標の領域は $[0,\ 1]^4$ となる。

補間関数 `Interpolation(`$\mathbf{x}$`,`T`)`はスカラー値 S を、セルの形状ごとに定義される、形状関数 $N(\mathbf{x})$ から計算する[14]。

$$S = \sum_{i=1}^{n_p} N_i(\mathbf{x}) S_i \tag{4.2}$$

ここで n_p は格子頂点の数であり、S_i は格子 T の頂点上に定義されたスカラー値である。

局所座標から大域座標への座標変換 `Mapping(`$\mathbf{x}$`,`T`)`は以下の式で計算される。

$$\mathbf{X} = \sum_{i=1}^{n_p} N_i(\mathbf{x}) \mathbf{X}_i^{node} \tag{4.3}$$

ここで $\mathbf{X}_i^{node}$ は格子 T の頂点の大域座標を意味する。

関数 `Normal(`$\mathbf{x}$`,`T`)`は、局所座標におけるスカラー場 $S(\mathbf{x})$ を用いて、大域座標におけるスカラー場 $S(\mathbf{X})$ の勾配を計算し、以下のように表される。

$$\nabla S(\mathbf{X}) = \mathbf{J}^{-1} \nabla S(\mathbf{x}) \tag{4.4}$$

ここで $\mathbf{J}$ は式(4.4)における変数変換から計算されるヤコビ行列を意味する。

4.1.3 実装

ある粒子位置 $\mathbf{X}$ に対応する色の値 c は、スカラー値 $S(\mathbf{X})$ を参照することで

伝達関数 tfunc(*S*)の色の関数 color()から計算される。色の値 c 、粒子位置 **X** 、そして法線ベクトル $\nabla S(\mathbf{X})$ は粒子を構成する情報として、関数 setParticle(c,**X**, $\nabla S(\mathbf{X})$)を用いてメモリ空間に格納される。メモリ空間上での粒子は、位置ベクトル(float*3=12 byte)、法線ベクトル(float*3=12 byte)、色ベクトル(byte*3=3 byte)から構成され、一つ当たり 18 byte の大きさを持つ。投影された粒子はフレームバッファ上で色ベクトル(3 byte)とデプス値(float*1=4 byte)を持ち、1 ピクセル当たり 7 byte の大きさを持つ。粒子生成の関数 ParticleGeneration の擬似コードを以下に示す。

Algorithm 1: ParticleGeneration ()

```
18:   calculate density function ρ;
19:   Transfer Function tfunc;
20:   for each cell Ti do
21:     Number of Particles Np;
22:     Np = NumOfParticles(Ti);
23:     Local Position x0, x';
24:     x0 = SamplingInLocal(Ti);
25:     Scalar Value s0, s';
26:     Color Value c;
27:     s0 = interpolation(x0,Ti);
28:     Global Position X;
29:     Normal n;
30:     while j < Np do
31:       x' = SamplingInLocal(Ti);
32:       s' = interpolation(x',Ti);
33:       Ratio of Density W = ρ(s')/ρ(s0);
34:       if W >= 1 then
35:         X = mapping(x',Ti);
36:         n = normal(x',Ti);
37:         c = tfunc(s').color();
38:         setParticle(c,X,n);
39:         x0 = x'
40:         s0 = s'
41:         j++
42:       end if
43:       else
44:         If W >= random() then
45:           X = mapping(x',Ti);
46:           n = normal(x',Ti);
47:           c = tfunc(x').color();
48:           setParticle(c,X,n);
49:           x0 = x'
```

```
50:           s0 = s'
51:           j++
52:         end if
53:       end else
54:     end while
55:   end for
```

この擬似コード **Algorithm 1** において、関数 NumOfParticles(T_i)は格子 T_i に対する生成粒子数を計算する。生成粒子数を計算するために、格子の重心位置の密度の値ρ_g と格子体積の積を計算する。この値は格子内部の密度分布を定数 ρ_g で近似した場合の式(3.15)の解と等しい。実数として計算された生成粒子数は、式(3.16)を用いて整数に変換される。また、1 より小さい生成粒子数の値は、1 つの粒子が生成される確率とみなして計算を行う。この計算を行うために、床関数 floor()と [0, 1] の範囲の乱数生成関数 random()を用いる。生成粒子数を計算する関数 NumOfParticles の擬似コードを以下に示す。

Algorithm 2: NumOfParticles (cell T_i)

```
1:    Integer Number of Particles Np;
2:    Volume v = Ti.getVolume();
3:    Local Position g;
4:    g = Ti.getCenterOfGravity();
5:    Real Number of Particles Rp;
6:    Rp = v * ρ(mapping(g, Ti));
7:    If Rp - floor(Rp) > random()then
8:      Np = floor(Rp) + 1
9:    end if
10:   else
11:     Np = floor(Rp)
12:   end else
13:   return Np
```

擬似コード **Algolithm 2** において、関数 getVolume()は着目している格子の体積の計算を行い、関数 getCenterOfGravity()は着目格子の重心位置 g を計算する。格子の重心位置は以下の式から計算される。

$$g = 1/n_p \sum_{i=1}^{n_p} \mathbf{X}_i^{node} \tag{4.5}$$

この節で示した生成粒子数の計算及び粒子生成手法は、格子毎に実行されると

言う特徴がある。一般的に不規則格子からなるボリュームデータは、有限要素法の計算の都合上、格子の頂点情報と、各格子を形成するための接続情報から構成される。そのためここで示した手法は、不規則格子ボリュームデータに即座に適用することができる。また、周辺の格子に依存する処理を含まないため、格子の隣接情報を生成する必要も無い。

4.2　リピート処理と画像詳細度制御

サブピクセル処理はサブピクセルの分割数を増やせば高い画質が得られる反面、フレームバッファのサイズが大きくなりメモリを圧迫すると言う問題点がある。この問題を解決するために、サブピクセル処理の代わりに、確率的に得られた画像の平均を計算するリピート処理を行う。粒子投影により得られた画像の輝度値を用意したバッファに逐次足しこみ、全ての投影処理が終わった後に平均値を計算する。粒子投影による輝度値を格納するバッファとその平均値を計算するためのバッファの二つのバッファのみが必要とされるため、サブピクセル処理よりもバッファの大きさを抑えられると考えられる。また、リピート処理はサブピクセル処理と組み合わせて画像生成を行うことも可能である。つまり、一回のリピート処理に用いられる画像を、サブピクセル処理を用いて生成し、その平均を計算することで最終画像を得る。

リピート処理を用いた画像生成を用いることで､容易に画像詳細度制御を実装できる。画像詳細度制御とは、データを動かす時には粗く、高速にレンダリングを行い、止めると詳細にレンダリングを行う。この手法によりユーザーは PBVR のレンダリング結果をよりインタラクティブに操作できるようになることが期待される。

以下ではサブピクセル処理について概観し、リピート処理について詳解する。そして、サブピクセル処理とリピート処理で得られる画像の画質についての指標となる関係式の導出を行う。

4.2.1　サブピクセル処理

サブピクセルレベルを L_S とする。画面上の各ピクセルは一様に L_S^2 個のサブ

ピクセルに分割されている。あるピクセルの最終的な輝度値 B^{total} は、そのピクセルに対応する全てのサブピクセルの輝度値 B^i を平均して得られる（図 4.2）。

$$B^{total}(L_S) = \sum_{i=1}^{L_S^2} \frac{B^i}{L_S^2} \tag{4.6}$$

サブピクセル処理による画質について、サブピクセルレベルを増加させていくに従い画像のゆらぎが減少していくことが確認されている。しかしサブピクセルレベルが L_S のとき、フレームバッファは画像解像度の L_S^2 倍のサイズが必要とされる。この性質は、高解像度の、もしくはサブピクセルレベルを上げて高画質な画像を得ようとする場合に、ビデオメモリを圧迫しボトルネックになる。

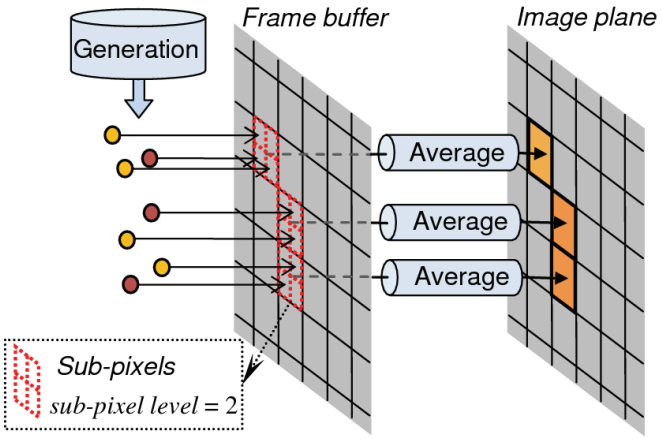


図 4.2　サブピクセル処理

4.2.2　リピート処理

PBVR で得られる画像のゆらぎの評価を行う実験により、画像平均によりサブピクセル処理と同様の高画質な画像を得られるという知見を得た。この性質を利用し、サブピクセルによる分割を行わずに粒子投影を繰り返し、得られた画像群の輝度値の平均を計算することで画質を改善する手法がリピート処理である（図 4.3）。平均処理に用いられる各画像は、同一の粒子半径、密度分布から得られたものであるので、ある確率場から独立に得られたサンプル画像だとみなせる。

リピート処理は各サンプル画像の色の値を平均することで最終的なピクセルの色の値を計算する。 i 番目のサンプル画像の輝度値を B^i とし、リピート処理の行われる回数を L_R とする。これをリピートレベルと呼ぶ。その時最終的なピクセルの輝度値は以下の式で計算される。

$$B^{total}(L_R)=\sum_{i=1}^{L_R}\frac{B^i}{L_R} \tag{4.7}$$

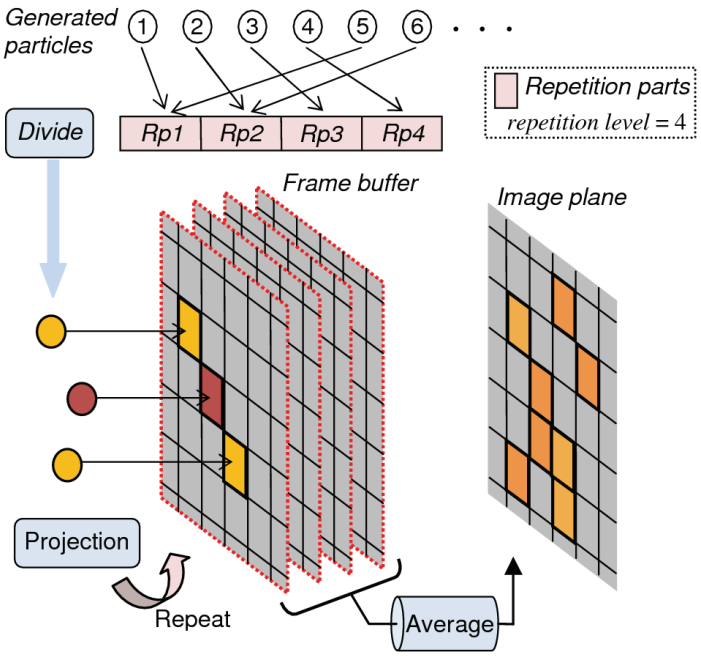


図 4.3　リピート処理

後述するリピート処理とサブピクセル処理の輝度値のゆらぎの解析から、サブピクセルレベル L_S とリピートレベル L_R の間には、$L_R = L_S^2$ の関係があることが分かる。この性質を用いて、リピート処理を用いる場合の粒子半径 r は、サブピクセル処理を計算する式(4.3)を変形して、以下のように求められる。

$$r=\frac{p_{obj}}{2\sqrt{L_R}} \tag{4.8}$$

ここで p_{obj} はオブジェクト空間中のピクセルの長さである。

リピート処理を行うためには、L_R 個の部分粒子群が必要になる。理想的には L_R 回粒子生成を繰り返して、部分粒子群を生成するべきであるが、粒子生成時間がボトルネックになる。これを回避するため、粒子半径 r で生成された粒子を L_R 個の部分粒子群に分割する。図 4.3 に示すように、生成された粒子は、配列の前から順に、周期的に部分粒子群の配列に格納される。生成した粒子数を N とするとき、i 番目（$i = 0, 1, \cdots, L_R\text{-}1$）の部分粒子群の粒子数 N_{sub} は以下の式を用いて計算される。

$$n_{sub}=\left\lfloor\frac{N}{L_R}\right\rfloor \tag{4.9}$$

$$i_{over} = N - L_R\, n_{sub} \tag{4.10}$$

$$N_{sub} = \begin{cases} n_{sub} + 1 & (i < i_{over}) \\ n_{sub} & (i \geq i_{over}) \end{cases} \tag{4.11}$$

部分粒子群は、リピートレベル L_R に従って確率統計的に生成された全生成粒子を均等に分割して生成されたものなので、全粒子の密度分布を維持することができる。部分粒子群は既存の PBVR の枠組みで投影が行われ、i 番目の部分粒子群から輝度値 B^i が得られる。投影毎に B^i は足し込まれ、最後に L_R で除算することで式(4.7)が計算される。

リピート処理を利用して、詳細度（Level of detail, LOD）制御を実装することができる。LOD 制御とはデータの可視化を効率的に行うために、データの特徴を極力保持したままデータ量や計算量を削減し、レンダリング時の負荷を減少させる手法である。ポリゴンデータに関する詳細度制御の手法は、曲率を測定し特徴的な形状を持つ部分を残してポリゴンの数を減らす技術 [42] [43] や、ポリゴンデータの表面上にマッピングされた色情報も考慮してポリゴンの数を減らす技術 [44] 等が挙げられる。またボリュームデータに関しては、大規模な四面体ボリュームを効率的に可視化するために、データの特徴を維持しながら四面体数を削減する技術 [43] が提案されている。本手法は部分粒子群の一部だけを用いた荒いレンダリングと、全粒子を用いたレンダリングを併用する。つまり、ユーザーがレンダリングされた対象を動かす場合に荒いレンダリングを行い、対象を止めて詳細に観察する場合に全生成粒子を用いた詳細なレンダリングを行う。こうしてレンダリング時の計算負荷を抑えることで、詳細度制御を実表する。荒いレンダリングに用いられる部分粒子群は全生成粒子と同等の密度分布を持つので、伝達関数で指定された色分布と不透明度をという特徴を維持することができる。

4.2.2.1　リピートレベルとサブピクセルレベルの関係式

サブピクセルレベルとリピートレベルが $L_R = L_S^2$ の関係を満たすとき、サブピクセル処理とリピート処理から同様の画質の画像が得られることを、簡単な粒子分布を持つモデルを用いて検証する。画質は輝度値の偏差を用いて評価する。そしてスカラー値が一定であるボリュームデータの内部に、輝度値 1 の粒子が密度 1 で一様分布しているモデルを仮定する。

まず、粒子投影を一回だけ行う場合について考える。先述したとおり、不透明度とは視線上に粒子が一つ以上存在する確率であると解釈できる。あるピクセル

に関する視線上に粒子がある場合、ピクセルの輝度値 B は 1 に、存在しない場合は 0 になる。粒子の存在確率は不透明度 α なので、ピクセルの輝度値を確率変数と考えると、B が 1 になる確率が α、B が 0 になる確率が $1-\alpha$ となる。このとき B の平均値 B_{Avg} は以下で計算される。

$$E(B)=B_{Avg}=1\cdot\alpha+0\cdot(1-\alpha)=\alpha \tag{4.12}$$

また、B の分散 B_{Var} は以下のように計算される。

$$Var(B)=B_{Var}=(1-B_{Avg})\alpha+(0-B_{Avg})(1-\alpha)=(1-\alpha)\alpha \tag{4.13}$$

ゆえに B の偏差 B_{Dev} は分散の平方根をとって以下のようになる。

$$B_{Dev}=\sqrt{(1-\alpha)\alpha} \tag{4.14}$$

次にリピート処理によって粒子投影が繰り返し行われる場合について考える。リピート処理に用いられる i 番目（$i=0, 1, \cdots, L_R-1$）の部分粒子群が投影されたピクセルの輝度値を B^i とする。式(4.12)(4.13)の結果は一回の粒子投影についてのものなので、B^i についても明らかに成り立つ。

$$E(B^i)=B_{Avg}=\alpha \tag{4.15}$$

$$Var(B^i)=B_{Var}=(1-\alpha)\alpha \tag{4.16}$$

部分粒子群の投影は、各処理において独立に実行されるため、各 B^i は独立な事象である。この時、共分散の値は 0 になる。

$$Cov(B^i,B^j)=0,\quad (i,j=0,1,\cdots,L_R-1) \tag{4.17}$$

輝度値の分散は式(4.7)を用いて、以下のように計算される。

$$B_{Var}^{total}(L_R)=Var\left(\sum_{i=0}^{L_R-1}\frac{B^i}{L_R}\right)=\frac{1}{{L_R}^2}\left\{\sum_{i=0}^{L_R-1}Var(B^i)+2\sum_{i,j,i<j}Cov(B^i,B^j)\right\}=\frac{B_{Var}}{L_R} \tag{4.18}$$

ゆえに、リピート処理で得られるピクセルの輝度値の偏差は以下のようになる。

$$B_{Dev.}^{total}(L_R)=\frac{B_{Dev.}}{\sqrt{L_R}}=\sqrt{\frac{(1-\alpha)\alpha}{L_R}} \tag{4.19}$$

式(4.19)から、L_R を増加させることで画像のゆらぎを低減できることが分かる。そしてこの式はリピート処理による LOD 制御の根拠になっている。すなわち、ユーザーが画像に対して許容できるゆらぎ（標準偏差）の値を閾値として与えることで、必要なリピートレベルの値を計算することができる。大きな閾値の値であるほど、必要なリピートレベルの値は小さくなる。

式(4.19)と式(3.26)を比較すると、同じ画質、すなわち同じ偏差の値のとき、明らかに以下が成り立つ。

$$L_R={L_S}^2 \tag{4.20}$$

式(4.20)から、サブピクセルレベル L_S と同等の画質をリピート処理で得るためには、リピートレベルを L_S^2 にすれば良いことが分かる。

4.3 実験結果と考察

提案手法の有効性を検証するため、粒子生成時間、レンダリング時間、画質、そしてメモリ使用量に関する実験を行う。

4.3.1 計算速度

提案手法を四面体からなる不規則格子に適用し、粒子生成時間とレンダリング時間の計測を行った。実験には Intel Core 2 Duo E8500 (3.17 GHz)の CPU、3GB の RAM、そしてグラフィックカードとして nVidia GeForce 9800 GT を搭載した CPU を用いる。実験に用いた不規則格子ボリュームデータを構成する頂点数と四面体数を表 4.1 に示す。これらのボリュームデータのうち、最大の四面体数は約 138 万である。

表 4.1　四面体数と節点数、生成粒子数、粒子生成時間、レンダリング速度

Data	# of tets.	# of nodes	# of particles	Sampling [msec]	Rendering [fps]
fighter	70,125	13,832	8,878,275	1906	14.0
blunt	222,414	40,948	13,121,216	2922	11.5
post	616,050	108,300	15,110,048	3281	11.0
aorta	1,386,882	248,992	10,782,231	2656	12.6

“fighter”データは、不規則四面体格子から構成されるボリュームデータであり、NASA Langley Research Center が行ったジェット戦闘機に関する風洞実験における空気の流れを数値シミュレーションした結果である。“blunt”データは、鈍頭フィンが取り付けられた平らなプレートの上を流れる空気を数値シミュレーションした結果のボリュームデータである。“post”データは非圧縮性流体である液体酸素の流れに関する数値シミュレーションの結果のボリュームデータである。平らな板の上に円柱が垂直に取り付けられており、液体酸素はその板の上を流れる。このシミュレーションはロケットエンジン内部の燃料の流れのモデルを用いている。スペースシャトルの打ち上げロケットのエンジンの内部にはこのような、燃料混合のための液体酸素の流れを阻害する円柱が多数存在する。“aorta”は、

大動脈瘤をもつ心臓の大動脈における血流の数値シミュレーションの結果のボリュームデータである。

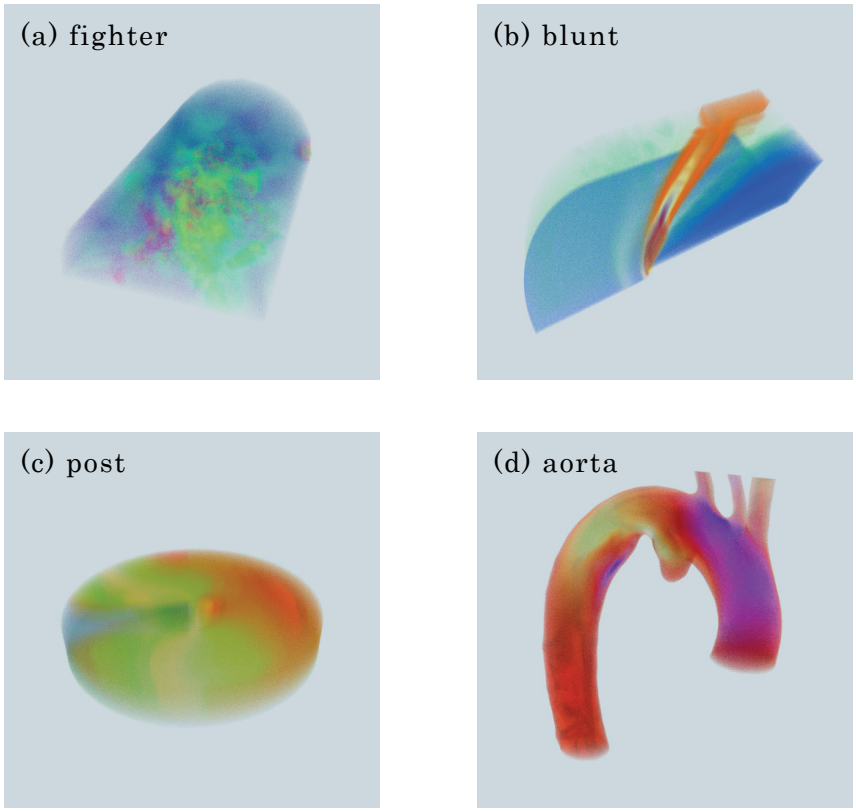


図 4.4　レンダリング画像

実験に用いる画像解像度は 512x512、リピートレベルは 144 である。表 4.1 に生成粒子数、粒子生成時間、そしてレンダリング時間を示す。また、図 4.4 にレンダリング結果示す。表 4.1 から、サンプリング時間はボリュームデータの格子数ではなく、生成粒子数に比例していることが確認できる。レンダリング速度に関して、明らかに生成粒子数が少ないほど速い傾向が見られる。PBVR はボリュームデータを粒子群に変換し、元の格子形状に関する情報を持っていない。そのため、レンダリング速度はボリュームデータの格子数や形状ではなく、生成粒子数に比例するのだと考えられる。生成粒子数は密度関数とスカラー場の関数の合成関数から計算されるため、同一のボリュームデータであっても不透明度の与え方によって生成粒子数およびレンダリング速度が大きく変わると考えられる。

4.3.2 画質の検証

定量的に画像の比較を行うために、リピート処理で生成される画像とサブピクセル処理で生成される画像に関する誤差の値を計算し、比較を行った。誤差を計算する上での評価基準にするために、ボリュームレンダリングにおいて広く用いられているレイキャスティング法で生成される画像を用いた。誤差の値は RGB 空間における 2 画像間の平均距離として定義した。画像の空白部分の寄与を無視するため、粒子の投影が行われていないピクセルは無視して計算を行った。誤差の計算式は、式(3.17)と同様である。実験には CPU が Intel Core 2 Duo E6750 (2.66 GHz)、RAM が 2.0 GB、そしてグラフィックカートカードとして nVidia GeForce 8500 GT を搭載したマシンを用いた。実験ではサブピクセルレベル L_S を 1 から 10 まで変化させた。サブピクセル処理と同等の画質を得るために、リピートレベル L_R を $L_R = L_S^2$ として変化させた。図 4.5 に結果を示す。リピート処理から得られた誤差とサブピクセル処理から得られた誤差が非常に接近した結果が得られた。図 4.6 にレイキャスティングで生成された画像、リピート処理で生成された画像、サブピクセル処理で生成された画像を示す。これらの結果から、リピート処理はサブピクセル処理と同等の画質の画像を生成できる手法だと考えられる。

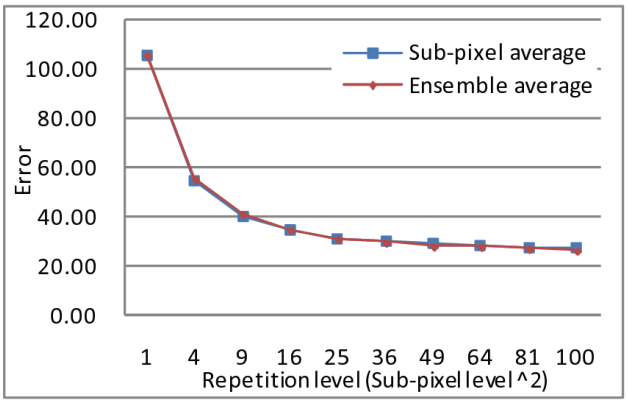
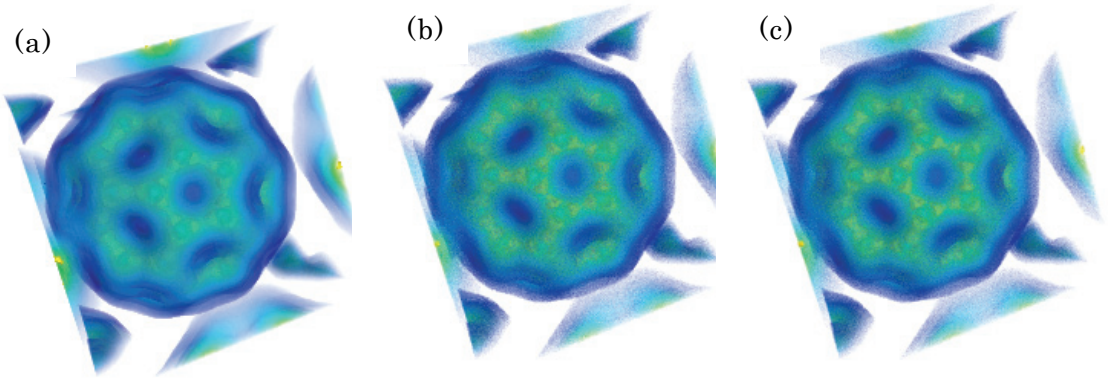


図 4.5　リピートレベルに対する誤差



(a)：レイキャスティングで生成された画像。

(b)：リピート処理で生成された画像。リピートレベル 100。

(c)：サブピクセル処理で生成された画像。サブピクセルレベル 10

図 4.6　レンダリング画像の比較

4.2 節では一様な粒子分布のモデルを用いて、ピクセルの輝度値の標準偏差がリピートレベルの平方根に逆比例することを示した。しかし実際のボリュームデータ上に生成された粒子は複雑な分布を形成する。そのため実データに対して生成された粒子群から得られた画像を用いて、リピートレベルに対する標準偏差の振る舞いを調査することが必要である。実験では、3 つのボリュームデータ“aorta”、“fighter”、“spx”を用いて、リピートレベルを 1 から 100 まで変化させた。各リピートレベルに対して、異なる乱数の種により生成された粒子群を用いて、20 枚の標本となる画像を生成し、各画素について標準偏差を計算した。図 4.7 のグラフは、横軸がリピートレベルを、縦軸が各画素の標準偏差を平均した値を意味する。粒子投影が行われていないピクセルは無視して平均を計算した。このグラフから、標準偏差の平均値が大体リピートレベルの平方根に逆比例して減少していることが確認できる。また、ボリュームデータが異なっても標準偏差の平均値はほぼ一致していることも確認できる。これらの結果から、4.2 節の結果が概ね一般的に成り立つと考えられる。

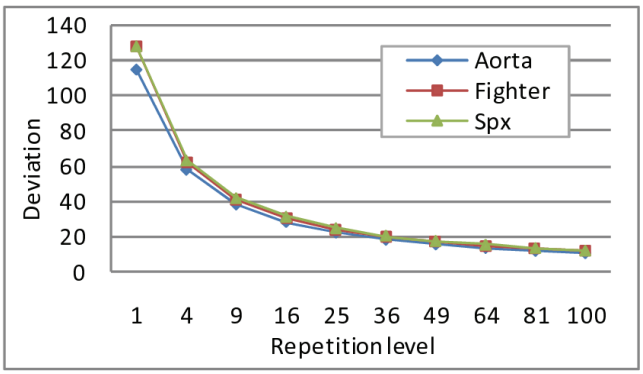
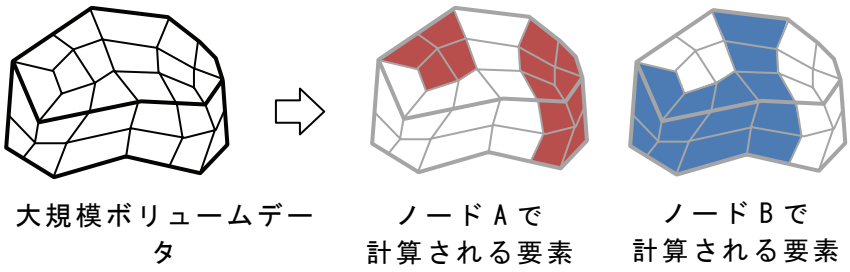


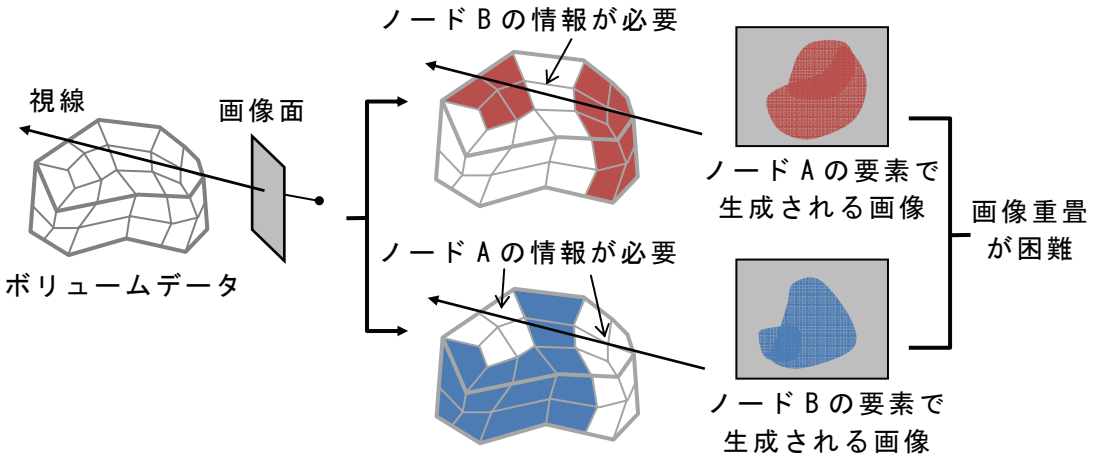
図 4.7　リピートレベルに対する標準偏差

4.3.3　大規模データへの適用例

分割された大規模不規則格子ボリュームデータは、ソート処理がボトルネックとなる顕著な例である。分散環境下で計算されたボリュームデータは、計算領域ごとに分割され、複数の部分ボリュームデータから構成される。このような、ボリュームデータを扱うための最も直接的な手段は格子を合体させることだが、大規模データを一ヶ所に集める事は、計算機の容量の観点で現実的ではない。分割したままでレンダリングを行うための一般的な戦略は、計算ノードごとに部分ボリュームをレンダリングした部分画像を生成し、それらを画像重畳することによって最終的なレンダリング画像を得ようとするものである。この方法は平面で分割された規則格子からなるボリュームデータの場合に上手くいくことが知られている [45] 。しかし部分ボリュームが非凸面体から構成される場合、ソーティング処理を行うことができず、この方法を適用することができない。また、画像重畳を用いず、視線や要素単位でのソート処理を行おうとする場合計算ノードへのアクセスが発生し、ボトルネックとなる（図 4.8）。このようなデータは、従来手法が実装されたボリュームレンダリングソフトウェアを用いても取り扱いが困難であり、断面コンターや等値面表示のような、面を用いた可視化が行われることが多かった。



(a) 領域ごとに分割された大規模ボリューム



(b) 分散レンダリング

図 4.8　大規模ボリュームデータの分散可視化

PBVR のソーティング不要という性質は、部分ボリュームから構成された大規模なボリュームデータを扱う際に、強力な利点となる。PBVR は、他の部分ボリュームを考慮することなく要素単位での粒子生成処理が可能である。そして生成した粒子を一つにまとめることで、部分ボリュームの形状と無関係にレンダリングを実行することができる（図 4.9)。

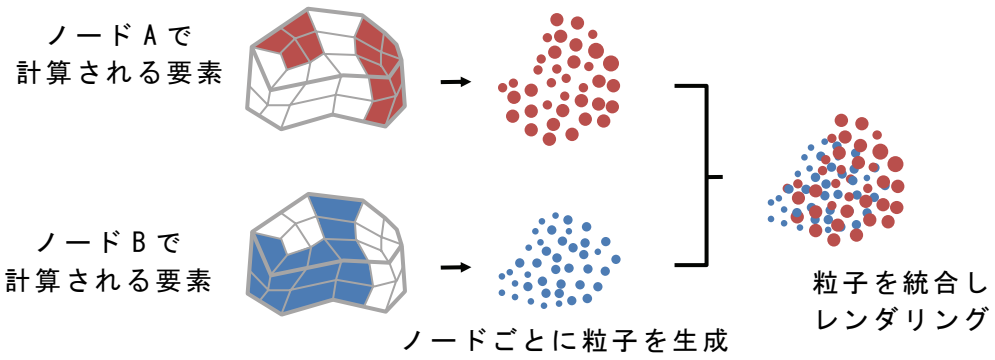


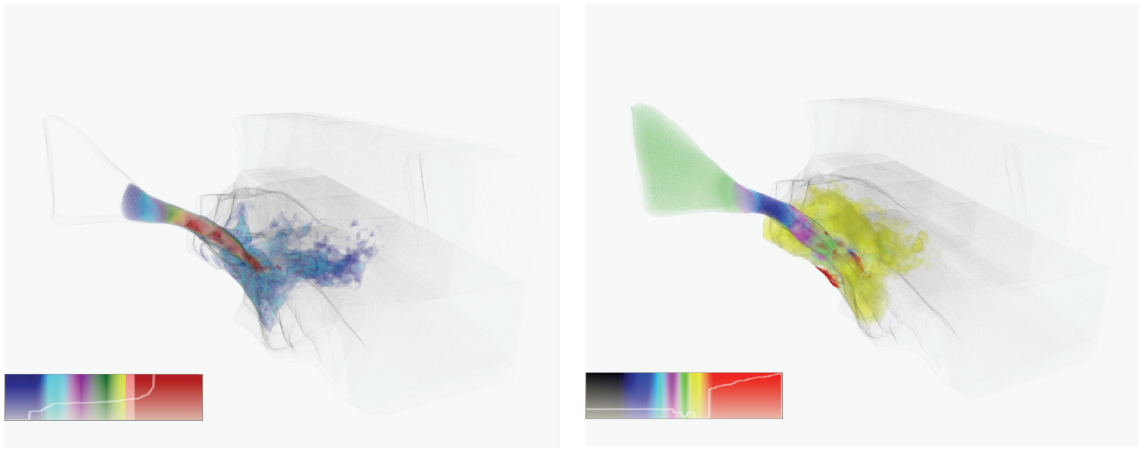
図 4.9　PBVR による分散可視化

提案手法を領域分割された大規模な CFD および CSM の結果のボリュームデータに適用し、その有効性を検証する。

4.3.3.1　口腔流体シミュレーション結果への適用

口腔内を流れる気流を計算し、摩擦による発音障害に関する解明および治療方法に関する研究が進められている。特に、口腔内気流解析において、その気流から発生する音源位置の推定は重要な問題の一つである。解析精度は格子の空間的密度、つまり要素数に依存し、加えて計算機性能の向上も伴い、ボリュームデータはますます大規模化している。この節では、LES（Large Eddy Simulation）に基づく流体解析ソフトウェア FrontFlow/Blue [46] を利用して計算された口腔気流シミュレーション結果のボリュームデータに適用した結果を示す。

シミュレーションの対象となる口腔内形状モデルは、コーンビーム CT を利用して得られた画像（画像解像度：512x512、スライス数：512）から等値面生成処理を行い、格子が生成されたものである。そして、その格子データから、264 万個の六面体要素からなる不規則格子型ボリュームデータが生成された。さらに、精度を向上させるために、稜線を 3 等分して 1 つの要素を 33 分割することにより、7,215 万個の六面体要素の大規模不規則格子型ボリュームデータが生成された。このボリュームデータ“Oral”は 71,449,236 の格子と 74,452,754 の頂点から構成され、16 の領域に分割されている。



(a) 速度の絶対値　　(b) 気圧

図 4.10　Oral データとその境界面のボリュームレンダ

図 4.10 は Oral データの可視化結果を示す。図 4.10(a)は気流の流れベクトルの絶対値のスカラー場を、(b)は気圧をボリュームレンダリングした結果である。同時に、各分割領域の境界面を灰色の曲面はとしてレンダリングした。これは境界面上に粒子を一様分布させ、ボリュームデータ内部に生成した粒子群と合体させることで、PBVR の枠組みの内部で実現した。構造格子に対するボリュームレンダリングと異なり、不規則格子のボリュームレンダリングを境界面の可視化と併用することで、ボリュームデータの形状と内部の値の分布の相対的な位置関係を把握できるようになる。

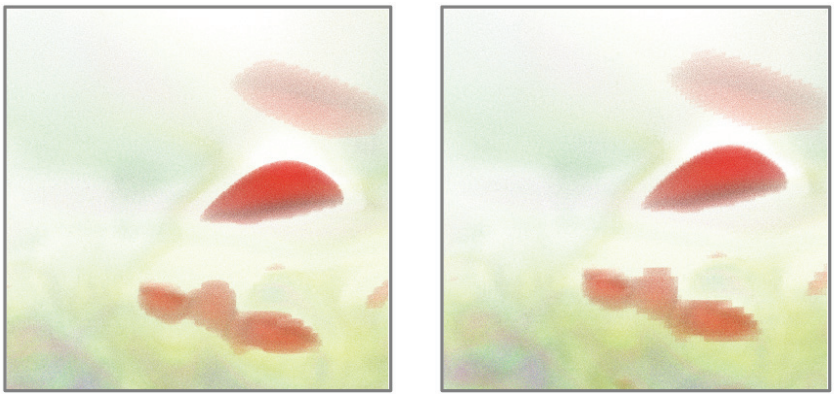
メモリ使用量の観点でリピート処理の有効性を検証するために、Oral データを用いてサブピクセル処理との比較を行った。同時にレンダリング速度も計測した。実験に用いた画像解像度は 512x512 である。実験ではサブピクセルレベルを 2 から 14 まで、それに対応してリピートレベルを 4 から 196 まで変化させた。表 2 に両手法によるフレームバッファのメモリ使用量[M byte]、レンダリング速度[fps]、そして生成粒子数[million]とそのメモリ使用量[M byte]を示す。表 4.2 から、サブピクセル処理のフレームバッファのメモリ使用量が、サブピクセルレベルの 2 乗に比例して増加していることが確認できる。また、サブピクセルレベル 6（リピートレベル 36）以下の場合に、サブピクセル処理がリピート処理よりも高速であることが確認できる。しかし、サブピクセルレベル 8（リピートレベル 64）の場合には、リピート処理がサブピクセル処理の速度を上回っていることが確認できる。そしてサブピクセルレベルがその値を超えると、実験に用いた環境では、サブピクセル処理に必要なフレームバッファを確保することができなかった。リ

ピート処理はリピートレベルの増加に伴いレンダリング速度が遅くなる反面、フレームバッファのメモリ使用量を抑えることに成功している。

表 4.2　サブピクセルレベルと対応するリピートレベルにおける
フレームバッファの大きさ、生成粒子数、レンダリング速度

Subpixel level	2	4	6	8	10	12	14
Frame buffer [M byte]	7.00	28.00	63.00	112.00	175.00	252.00	343.00
fps	172.86	52.90	21.45	8.22	Na.	Na.	Na.
Repetition level	4	16	36	64	100	144	196
Frame buffer [M byte]	2.50	2.50	2.50	2.50	2.50	2.50	2.50
fps	150.29	41.50	18.93	10.10	6.45	4.01	2.69
Num. of particles [million]	0.17	0.69	1.55	2.76	4.32	6.22	8.46
Particle mem. [M byte]	2.96	11.83	26.66	47.44	74.17	106.71	145.21

粒子生成手法の有効性を検証するために、メトロポリスサンプリングと一様分布による粒子生成時間と、それらにより生成された画像の比較を行った。図 4.11 は両手法により生成された、Oral データの拡大画像である。表 3 に粒子生成時間と生成粒子数を示す。図 4.11(b)より、一様分布による画像の赤色の部分にブロッキーノイズが生じていることが確認できる。そして、図 4.11(a)のメトロポリスサンプリングによる画像では、その部分のブロッキーノイズが明らかに軽減されていることが確認できる。表 4.3 より、一様分布の粒子生成がメトロポリスサンプリングよりも高速であることがわかる。



(a) メトロポリスサンプリング　　(b) 一様分布

図 4.11　Oral データの拡大画像による画質の比較

表 4.3　Oral データに対する粒子生成時間と生成粒子数

	Metropolis	Uniform
Sampling time [sec.]	25.75	24.07
Num. of particles [million]	31.89	

4.3.3.2　構造解析シミュレーション結果への適用

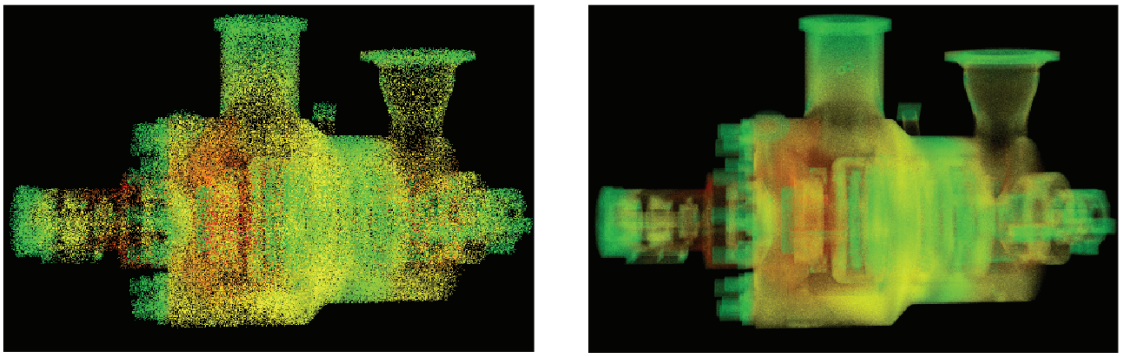
並列有限要素法支援ミドルウェア HEC-MW[11]を基盤とした大規模並列構造解析システム FrontSTR [46] を利用した原子炉給水ポンプの構造解析結果のボリュームデータに対して実験を行った。

ボリュームデータ“Pump”は、PC クラスタで計算された、一億自由度の自重による静的弾性解析の計算結果である。Pump データは 32 の領域に分割され、26,289,770 の四面体二次要素による格子で構成され、36,728,129 の頂点を持つ。従来では、断面コンター図や表面幾何形状データに対してミゼス応力などのデータ値をマッピングし可視化を行っていたが、ボリュームレンダリングを行うことによって、ポンプ内部のデータ値の空間的な変化を解析することが可能となる。

提案手法を Pump データに適用し、LOD 制御を行った。有効性を検証するために、荒いレンダリングと詳細なレンダリングの画質とレンダリング速度を比較した。実験ではリピートレベル 100 で粒子生成を行った。表 4.4 に生成粒子数と粒子生成時間を示す。図 4.12 に LOD 制御を用いたレンダリング結果を示す。図 4.12(a)はリピートレベル 2 による荒いレンダリングの画像を、図 4.12(b)はリピートレベル 100 による詳細なレンダリングの画像をそして表 4.5 には両者のレンダリング速度を示す。表 4.5 から、荒いレンダリング時にはインタラクティブフレームレートが実現されていることが確認できる。また図 4.12 から、荒いレンダリングの画像が、画質を落としながらも、不透明度と色の分布という特徴を維持していることが確認できる。

表 4.4　Pump データに対する粒子生成時間と生成粒子数

Repetition level	Sampling time [sec.]	Num. of particles
100	47.65	13,097,195



(a) Repetition level 2 to move　　(b) Repetition level 100 to stop

図 4.12　LOD を用いた Pump データのレンダリング画像

表 4.5　LOD を用いた Pump データのレンダリング速度

	Coarse rendering	Fine rendering
Repetition level	2	100
fps	93.73	1.58

4.4　まとめ

従来の PBVR は四面体以外の不規則格子に対して粒子のサンプリングを行うことができなかった。この問題を解決するために、粒子密度推定式を用いて PBVR を不規則格子に適用するための粒子分布生成手法を提案した。この手法は有限要素法で用いられるアイソパラメトリック要素に対する補間関数と形状関数を利用して、局所座標内部でサンプリングを行い、大域座標へマッピングする。そして、格子内部での粒子密度分布を生成するために、局所座標内部でメトロポリスサンプリングを行った。

PBVR はサブピクセルの分割数を増やせば高い画質が得られる反面、フレームバッファのサイズが大きくなりメモリを圧迫すると言う問題点があった。画像生成時の PBVR のメモリ効率を改善するため、画像重畳を用いてレンダリング画像を生成する、リピート処理を提案した。リピート処理はサブピクセル処理の代わりに、確率的に得られた画像の平均を計算することで画質が向上するという、PBVR の画像のゆらぎの解析から得られた結果を用いている。更に、リピート処理を応用して画像詳細度制御を実現した。すなわち、データを動かす時には粗く、

高速にレンダリングを行い、止めると詳細にレンダリングを行うという技術である。

リピート処理とサブピクセル処理の画像を比較した結果、同等の画質の画像が得られることがわかった。また使用したフレームバッファの大きさを計測した結果、リピート処理によりメモリ使用量を抑えられることが確認できた。

提案手法を有限要素法で計算された実データに適用し、従来手法ではボリュームレンダリングが困難だった大規模不規則格子ボリュームデータ（領域分割された 7200 万の六面体要素や、2600 万の四面体二次要素から構成されたボリュームデータ）をボリュームレンダリングすることに成功した。また、粒子生成に一様分布を用いた以前の PBVR と画質を比較した結果、メトロポリスサンプリングの効果により、ブロッキーノイズが軽減されていることが確認できた。

有限要素法等の不規則格子を用いる数値シミュレーションは、高帯域幅のスーパーコンピュータや PC のクラスタで並列計算されるようになった。出力結果は大規模化し、それらを可視化するために同等の大きさの並列計算リソースが必要とされるようになった。しかし PBVR の並列化効率の高さは、こういった問題に対する有望な解決法となる可能性がある。

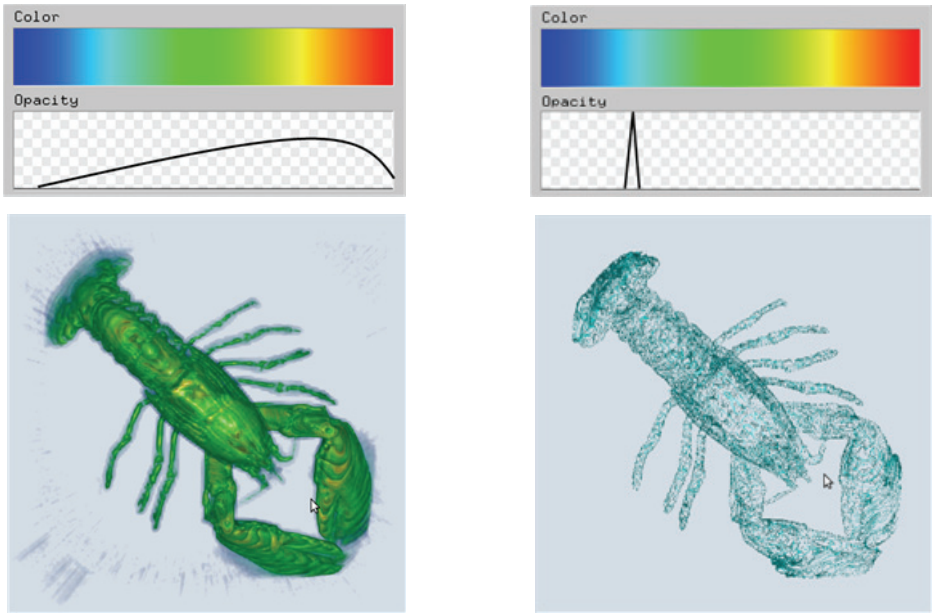
提案手法のレンダリング速度は生成粒子数に影響される。そして生成粒子数はボリュームレンダリングに用いるパラメータの値を変更することで変化する。伝達関数やオブジェクト座標上でのピクセルの大きさを変更した場合、不透明度や粒子半径が変化するため、密度関数が再計算される。この時、生成粒子数がシステムのメモリ容量を超えて、粒子を格納できなくなる可能性がある。例えば画像面の解像度が高い場合、オブジェクト座標上での粒子半径が相対的に小さくなり、生成粒子数が増大する。こうした場合でも、PBVR は格子のストリーミングを用いてレンダリングを行うことができると予想される。この手法は粒子群をメインメモリに格納せずに、ハードディスク上の不規則格子ボリュームデータの各格子をメモリに送信し、その格子に対してのみ粒子生成を行い即座に投影を行って画像を生成する。昨今、高解像度液晶の開発や、タイルドディスプレイ技術の発達に伴い、高解像度環境下でのボリュームレンダリングが必要とされる局面が増えると考えられる。そのため、どのような生成粒子数にも対応可能な PBVR のレンダリング技術の開発が期待される。

第5章　四面体向けレイヤードサンプリングと粒子半径の調整

ボリュームレンダリング画像の見栄えを決定する伝達関数の編集において、ユーザーは急峻に変化する不透明度を設定することがある。伝達関数の編集の方法はユーザーの見たい情報に従って様々な方針が採られる。そして急峻な伝達関数は、ユーザーが見たい情報を全体から一部へ絞っていく過程でしばしば出表する。初めにユーザーはスカラー場全体の分布を知るために、緩やかに変化する伝達関数を与えて可視化を行う（図 5.1(a)）。その画像を見たユーザーがある特定の範囲のスカラー値を強調したいと考えた場合、その範囲のスカラー値には高い不透明度を与える。そして、そうでないスカラー値の不透明度は下げるか、或いは 0 にする（図 5.1(b)）。この着目したスカラー値の範囲が小さいほど不透明度関数は急峻に変化する。

レイキャスティングや四面体投影法等の従来手法によるボリュームレンダリングは、高速に計算を実行するために、視線上の各サンプリング区間において密度が一定になる仮定を置いている。そのため、サンプリング間隔を十分に細かく与えるか不透明度の変化が緩やかであれば、スカラー場の特徴を損なうこと無く画像を生成できる。しかし急峻に変化する伝達関数が与えられた場合、視線上での不透明度関数の変化がサンプリング間隔よりも小さくなり、画像にアーティファクトが生じることがある（図 5.1(b)）。

また PBVR においても、伝達関数が急峻に変化する場合、発生粒子数の計算と粒子生成を正確に行うことが困難であり、画像にアーティファクトを生じた。PBVR による画像は、粒子密度推定法により計算されるボリュームデータ内部の粒子密度分布に依存するため、適切な生成粒子数の計算が必要となる。しかし従来の PBVR は生成粒子数の計算に、格子の密度に関する一点積分を用いていた。そのため急峻な伝達関数に対してサンプリングが失敗し、積分精度が下がっていた。本研究では、四面体格子に適用可能な高品位な粒子生成手法である、四面体向けのレイヤードサンプリング法を提案した。



（a）緩やかな伝達関数　　（b）急峻な伝達関数

図 5.1　レイキャスティングによる画像の比較

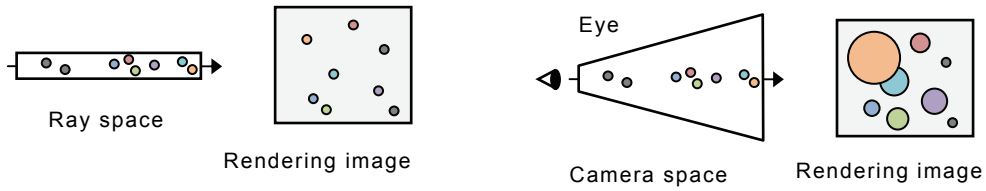
5.1　粒子拡大による画像生成

密度発光モデルでは、視線は円筒で近似される。しかし本来は、視線は視点から放射状に広がりので、円錐状の視線を考えるのが自然である。ゆえに、視線上の各区間に含まれる粒子の数は距離の 2 乗に比例して増加する。また密度発光モデルでは、視線上のある区間から到達する粒子の輝度値が距離と無関係に一定値に固定される。しかし、粒子から発生した光は、点光源であるため、距離の 2 乗に反比例して減衰していくのが自然である。両者を考慮すると、円錐状の視線空間の中では、距離に比例する粒子密度の増加と、反比例する輝度値の減衰が打ち消しあい、結果として合計の粒子による輝度値は距離と無関係になる。つまり、視点からの距離に関係なく、円筒状の領域から届く輝度値が一定値と仮定するのが妥当である。

円筒状の領域において到達する輝度値が一定であるということは、円錐状の領域においては、視点の近くほど輝度値が増加せねばならない。（視点がボリュームデータから十分に遠く、平行投影に近似できる場合には、両者は一致する。）PBVR は粒子半径を距離と無関係に一定と仮定し、画像面上のサブピクセルと同一のサ

イズで描かれていた（図 5.2(a)）。そのため視点の近くでは粒子密度が不足し、輝度値が減少してしまう。これは、視点からの距離に応じて適応的に粒子生成を行うことで回避されるが、インタラクティブなレンダリングを実表する意味では負の要因になる。

これを回避するために、粒子半径を視点からの距離に応じて変える手法を提案する（図 5.2(b)）。PBVR は生成した粒子を投影するため、円筒のモデルではなく円錐のモデルを考慮するべきである。PBVR において、粒子はボリュームデータの存在するオブジェクト座標中に生成されるため、距離の 2 乗に比例した粒子密度の増加が実現されている。しかし、距離の 2 乗に反比例した粒子の輝度値の減衰が、言い換えると、視点からの距離の逆数の 2 乗に比例した輝度値の増加が実現されていない。輝度値は単位面積当たりの光の強度であるので、投影された粒子像の輝度値を増加させるために、視点からの距離の逆数の 2 乗に比例するように粒子像の面積を増加させる。投影像を円で描くとすると、その面積は、粒子半径 r を用いて、πr^2 になるので、粒子半径を視点からの距離の逆数に比例させればよいことがわかる。



(a)視点からの距離に無関係な粒子半径　　(b) 視点からの距離に応じた粒子半径

図 5.2　円筒状の視線空間と円錐状の視線空間

PBVR において、粒子の直径はボリュームデータの重心座標におけるピクセルの長さとしている。オブジェクト座標において、任意の位置にある粒子と視点との距離を d、ボリュームデータの重心位置と視点との距離を d_{cen} 、重心位置における粒子半径を r_{cen} とする。このとき、任意の位置にある粒子の半径 r は以下の式で計算される。

$$r = \frac{d_{cen}}{d} r_{cen} \tag{5.1}$$

この式は d が 0、すなわち粒子位置と視点位置が等しい時に発散するが、粒子は描画の際に視錐台でカリングされるため、実際に d が 0 になることはない。

粒子の位置が重心位置よりも遠く、r が 1 よりも小さくなる場合、乱数を用い

て d_{cen}/d の確率で描画を行う。

粒子拡大を用いた PBVR の粒子は、スプラッティング法において用いられる再構成核を完全に不透明に定めものとよく似ている。加えて、スプラッティング法は不規則格子のボリュームレンダリングに用いられることがあるため、しばしば PBVR と混同される。しかしスプラッティング法が輝度値方程式を用いたボリュームレンダリング法の一種であるのに対し、PBVR は密度発光モデルに立ち返った手法である。以下では大規模不規則格子への適用と言う観点で、PBVR とスプラッティング法の違いを述べる。

5.1.1 PBVR とスプラッティング法

スプラッティング法と PBVR は、両者ともに粒子に基づいたレンダリングプリミティブを用いているという点において、よく似た手法である。スプラッティング法は粒子を少なくとも、ボリュームデータを構成するセルと同じ数だけ生成する必要がある手法である。そのため、粒子数はセル数に比例して増加する。それに対して、PBVR で生成される粒子数はセル数に比例しない。PBVR では、伝達関数の不透明度から計算される、粒子密度推定関数を参照しながら粒子生成を行う。つまり PBVR の生成粒子数はユーザー指定である伝達関数の不透明度に依存して決定される。そのため、PBVR では粒子が一つも生成されないセルが生じ得る。この特徴は、とりわけ大規模なボリュームデータを扱う際に、重要になる。例えば、セル分割を実行することでボリュームデータのセル数を増加させても、生成粒子数はほぼ一定になる。また、例えば高い不透明度が指定された場合、生成粒子数は増加する。生成粒子数を減少させるためには、伝達関数の不透明度を最適化したり、粒子半径を拡大したりする必要がある。

スプラッティング法と PBVR には、粒子の性質に関する重要な性質の違いがある。スプラッティング法は伝達関数を参照することで計算される不透明度を持った、半透明な粒子を用いる。そして半透明な粒子をレンダリングするためには、各視線方向に沿った粒子のソーティングが必要とされる。しかし PBVR は、密度発光モデルが仮定する完全に不透明な粒子を用いるため、粒子のソーティングが必要とされない。その代わりに、Z-バッファアルゴリズムを用いて、隠れた粒子の投影像の消去を行うだけである。この性質は PBVR の画像描画時間が明らかにスプラッティング法のそれを上回ることを意味している。

5.2　四面体向けレイヤードサンプリング

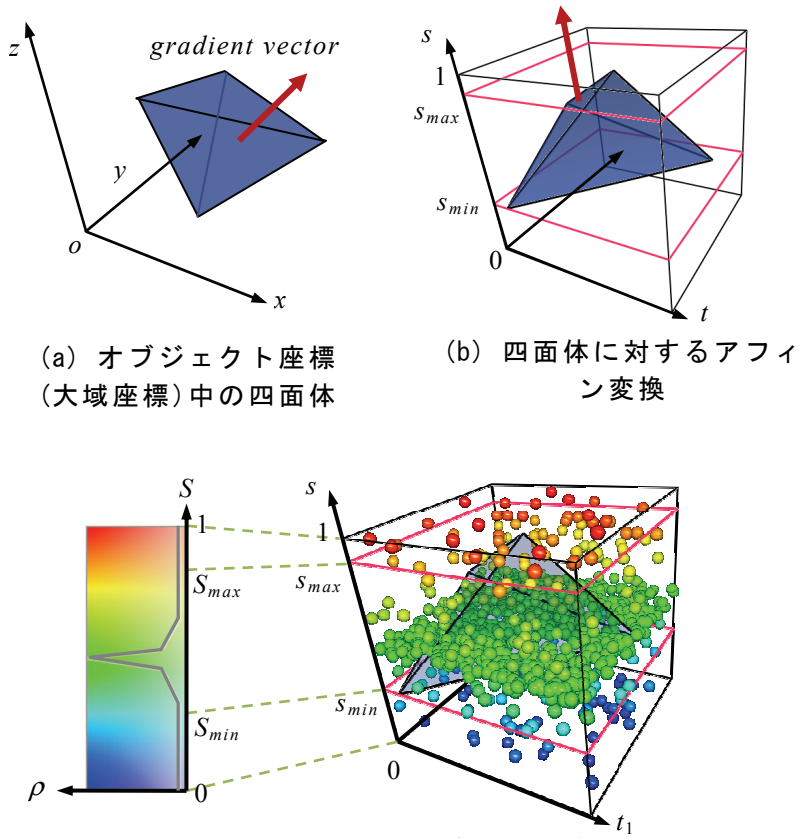
四面体格子向けのレイヤードサンプリングは、密度分布に従うサンプリング点群を計算用の空間にあらかじめ生成しておき、各四面体をそこに挿入することによって粒子のサンプリングを行う手法である。また、計算用の空間からサンプリングされた粒子を用いて、積分値を計算するための公式を導出した。

レイヤードサンプリングでは、区分線形伝達関数が仮定される。ユーザーが伝達関数を編集する方法として、自由曲線や、ガウス関数の平均と分散を指定するタイプ等様々あるが、制御点を指定して、その間の値を線形で補間する方法が区分線形伝達関数である。区分線形伝達関数は、区分線形色関数と区分線形不透明度関数から構成される。制御点は、スカラー値と色・不透明度の値の組で表現される。レイヤードサンプリングでは制御点と制御点の間の、各区間で粒子生成を行う。

スカラー値が頂点に定義された四面体格子の内部では、一般的に内部のスカラー値は線形に補間される。レイヤードサンプリングはこの性質を利用して効率的に粒子生成を行う。線形補間の仮定より、スカラー値の勾配は格子に対して一定になる。そして格子内部のスカラー場は、伝達関数の各区分に対応して、各層（レイヤー）に分割される。各層内部で粒子は勾配ベクトルの方向に線形に分布する。

この性質から、効率的なサンプリング手法のヒントが得られる。まず、立方体の領域を用意し、その領域の軸の一つをスカラー値とする。この立方体の領域を以降レギュラーボックスと呼ぶ。そして、レギュラーボックスの内部で、スカラー軸に対して垂直な各層に対して粒子を生成する。この粒子群を事前生成粒子群と呼ぶ。

一度レギュラーボックス内部に粒子生成を行えば、各四面体格子の粒子生成の速度を向上させることができる。粒子生成のために、四面体格子がレギュラーボックスに収まり、かつ勾配ベクトルがレギュラーボックスのスカラー軸と平行になるように、四面体格子に対してアフィン変換を適用する。そして、四面体格子内部に含まれる事前生成粒子群が、最終的な生成粒子の候補になる。



(a) オブジェクト座標（大域座標）中の四面体

(b) 四面体に対するアフィン変換

(c) 区分線形密度関数 $\rho(S)$

(d) レギュラーボックス内の事前生成粒子群と四面体

図 5.3　四面体向けレイヤードサンプリングの概要

この方法は、例え急峻に変化する伝達関数が与えられても、それを反映した密度分布が生成されるため、画質の劣化が抑えられると期待される。

レイヤードサンプリングは 2 段階の処理から構成される。第一段階では、伝達関数を用いてレギュラーボックス内部に粒子を事前生成する。第二段階では、事前生成粒子群を用いて四面体格子内部に粒子生成を行う。

5.2.1　事前生成粒子群

この説ではレギュラーボックスについて、そして事前生成粒子群を生成する方法について述べる。図 5.3(a)(b)に、アフィン変換により四面体格子がレギュラーボックス内部に挿入される様子を示す。スカラー値の範囲は、[0,1] に正規化されているものとする。四面体格子内部は線形のスカラー場であるため、スカラーの最大値と最小値は、四面体頂点上に存在する。勾配方向がスカラー軸と平行にな

るようにアフィン変換を行った場合､最小値を持つ頂点がスカラー軸の 0 方向に、最大値を持つ頂点がスカラー軸の 1 の方向を向く。

区分線形関数の例を図 5.3(c)に、それに従って生成された事前生成粒子群を図 5.3(d)に示す。事前生成粒子群は区分線形関数で与えられた、スカラー値の各区間で生成される。スカラー値の各区間で粒子分布は線形なため、生成が容易である。本手法では棄却法を用いて、線形の密度分布を生成する。粒子の事前生成を行うために、ユーザー指定のパラメータとして、粒子数 N を与える。スカラー値の各区間における粒子数を計算するために、レギュラーボックス内部の密度場を積分する。得られた積分値は事前生成粒子数 N をスカラー値の区間で分配するための比率となる。

(t_1, t_2, s)はレギュラーボックスの存在する空間の座標軸を意味する。s はスカラー値の S と同じ値（つまり(S = s)）である。t_1-, t_2-, s-座標において、レギュラーボックスは$[0,1]^3$ の領域である。レギュラーボックス内部で、事前生成粒子群は t_1, t_2 軸に関して一様分布し、s 軸方向にのみ線形分布する。

この手法では密度を計算する関数として、区分線形密度関数 $\rho(S)$ を用いる。この関数は区分線形不透明度 $\alpha(S)$ の各制御点について粒子密度値を計算し得られる。制御点としてスカラー値と密度値の組を持ち、制御点の間は線形補間により計算される。スカラー値の区間$[S_i, S_{i+1}]$（i は制御点に与えられるインデックス）に関する線形補間は以下の式で計算される。

$$\rho(S)=\frac{\rho_{i+1}-\rho_i}{S_{i+1}-S_i}S+\frac{\rho_i S_{i+1}-\rho_{i+1}S_i}{S_{i+1}-S_i}, \tag{5.2}$$

ここで ρ_i とρ_{i+1} は S_i と S_{i+1} に対応する密度値である。

スカラー値の区間$[S_i, S_{i+1}]$に対して、生成すべき粒子数は $devide_i$ は、以下のように計算される。

$$m_i=\int_{S_i}^{S_{i+1}}\rho(S)\,ds=\frac{1}{2}(\rho_i+\rho_{i+1})(S_{i+1}-S_i) \tag{5.3}$$

$$M=\int_0^1\rho(S)\,ds=\sum_i m_i \tag{5.4}$$

$$devide_i=\frac{m_i}{M}N \tag{5.5}$$

ここで m_i / M は N を分配する割合になっている。

5.2.2 事前生成粒子群を利用した粒子生成

粒子生成は複数の処理から構成される。それらは、四面体格子のレギュラーボックスへのアフィン変換を計算する処理（*Transformation*）、四面体格子をレギュラーボックスへ挿入し、事前生成粒子群との包含関係を計算する処理（*Insertion*）、四面体格子内部に含まれていた粒子を用いて、着目格子中に生成する粒子数 N_{tet} を体積積分により計算する処理（*Integration*）、着目格子に含まれていた粒子から N_{tet} 個の粒子を選び出す処理（*Selection*）、そしてその粒子の座標を逆変換して、もとの空間に戻す処理（*Inverse Transformation*）である。

アフィン変換は、2 つの行列 **L** と **A** から構成され、*Transformation* だけでなく、*Integration* や *Inverse Transformation* にも用いられる。行列 **L** は四面体格子の勾配ベクトルをオブジェクト座標の z 軸方向に向ける回転行列であり、この変換で勾配ベクトルはレギュラーボックスの s 軸と平行になる。行列 **A** は、四面体格子がレギュラーボックスに収まるようにスケーリングと平行移動を行う。ここで、*Tet* はもとの四面体格子を、*Tet'* は **L** で変換された四面体格子を、*Tet''* はレギュラーボックスに挿入された四面体格子を意味する。つまり、*Tet''* = **A** *Tet'* = **A L** *Tet* の関係がある。

行列 **L** は正規化された勾配ベクトル g とベクトル u から構成される。ここで u は g に対して線形独立な任意のベクトルである。u を計算するために、g の各成分の絶対値を比較する。そして、最小の絶対値の成分を 1、その他の成分を 0 とする。これらのベクトルを用いて、正規直交基底 $\{l_1, l_2, g\}$ を構成するベクトル l_1 と l_2 を計算する。

$$l_1 = u \times g, \quad l_2 = g \times (u \times g) \tag{5.6}$$

これらのベクトルを用いて、行列 **L** は以下のように構成される。

$$\mathbf{L} = \begin{bmatrix} l_1 & l_2 & g & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^T \tag{5.7}$$

この行列は、直交行列になっているので、明らかに g を s 軸と平行に、つまり $(0,0,1)^T$ に変換する。

行列 **A** は *Tet'* のバウンディングボックスがレギュラーボックスと重なるようにスケーリングと平行移動を行う。*Tet'* がある座標空間を (x', y', z') で記述する。*Tet'* のバウンディングボックスを x', y', z' 空間と直交するように定める。*Tet'* のバウンディングボックスの領域を $[x'_{\min}, x'_{\max}]$ x $[y'_{\min}, y'_{\max}]$ x $[z'_{\min}, z'_{\max},]$ と

する。レギュラーボックスの領域は$[0,1]^3$である。$S_{\min}$ と $S_{\max}$ は *Tet'* のスカラーの最小値と最大値である。

*x'*と *y'*方向のスケーリングと平行移動は、バウンディングボックスがレギュラーボックスと完全に重なるように行う。つまり、領域 $[x'_{\min}, x'_{\max}]$ x $[y'_{\min}, y'_{\max}]$ が領域$[0,1]^2$に重なるように変換する。

レギュラーボックスの *s* 軸はスカラー値と等しいので、*z'*方向のスケーリングと平行移動は、バウンディングボックスの *z'* 座標がスカラー値 *S* と一致するように行う。つまり、領域 $[z'_{\min}, z'_{\max}]$ が領域 $[S_{\min}, S_{\max}]$ に重なるように変換する。

これらの操作をまとめて、行列 **A** は以下のように構成される。

$$\mathbf{A}=\begin{bmatrix} a_1 & 0 & 0 & b_1 \\ 0 & a_2 & 0 & b_2 \\ 0 & 0 & a_3 & b_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.8}$$

$$a_1=\frac{1}{x'_{\max}-x'_{\min}} \quad b_1=-a_1 x'_{\min} \tag{5.9}$$

$$a_2=\frac{1}{y'_{\max}-y'_{\min}} \quad b_2=-a_2 y'_{\min} \tag{5.10}$$

$$a_3=\frac{S_{\max}-S_{\min}}{z'_{\max}-z'_{\min}} \quad b_3=-a_3 z'_{\min}+S_{\min} \tag{5.11}$$

5.2.3　レイヤードサンプリングにおける体積積分計算式

行列 **L** と **A** から *Tet''* の頂点座標を計算し、事前生成粒子群との包含関係を調べる。*Tet''* の内部にあった粒子の個数 N_{in} を求め、生成粒子数の計算に用いる。また *Tet''* の内部にあった粒子インデックスを保存し、後の逆変換で用いる。

四面体格子の生成粒子数 N_{tet} は密度関数の体積積分で計算される。レイヤードサンプリングにおいて、その値は以下の式により数値的に計算される。

$$N_{tet}=M\frac{1}{|a_1 a_2 a_3|}\frac{N_{in}}{N} \tag{5.12}$$

式(5.12)は密度関数$\rho(S)$の体積積分において、アフィン変換による変数変換を行うことで得られた。ここで $p(S)$ は $\rho(S)$ に対応した確率密度関数とする。**r** はレギュラーボックス内部の空間 $(t_1, t_2, s)^{\mathrm{T}}$ の位置ベクトルとする。そして **x** はオブジェクト座標 $(x, y, z)^{\mathrm{T}}$ の位置ベクトルとする。

A と **L** の定義から、**r** と **x** の関係は以下の式で表される。

$$\mathbf{AL}\begin{bmatrix}\mathbf{x}\\1\end{bmatrix}=\begin{bmatrix}\mathbf{r}\\1\end{bmatrix}. \tag{5.13}$$

Tet 内部の線形補間によるスカラー場は、係数 *a*、*b*、*c*、*d* を用いて以下のように表される。

$$S=\begin{bmatrix}a & b & c & d\end{bmatrix}\begin{bmatrix}\mathbf{x}\\1\end{bmatrix}. \tag{5.14}$$

式(5.13)に式(5.14)を代入して、以下の式が得られる。

$$S=\begin{bmatrix}a & b & c & d\end{bmatrix}\begin{bmatrix}\mathbf{x}\\1\end{bmatrix}=\begin{bmatrix}a & b & c & d\end{bmatrix}\mathbf{L}^{-1}\mathbf{A}^{-1}\begin{bmatrix}\mathbf{r}\\1\end{bmatrix}=s. \tag{5.15}$$

レギュラーボックス内部の領域 $[0, 1]^3$ における $\rho(S)$ の体積積分の値は、明らかに *M* になる。

$$\iiint_{[0,1]^3}\rho(S)dt_1dt_2ds=\int_0^1\rho(s)ds=M \tag{5.16}$$

式(5.16)と確率密度関数の規格化条件より、*p*(*S*)とρ (*S*)の間には以下の関係がある。

$$M\,p(S)=\rho(S) \tag{5.17}$$

確率密度関数の定義から、領域 *Tet''*における *p*(*S*)の体積積分は以下のように計算される。

$$\iiint_{Tet''}p(S)dt_1dt_2ds=\frac{N_{in}}{N}, \tag{5.18}$$

式(5.13)－(5.18)の結果を用いて、領域 *Tet* におけるρ (*S*) の体積積分は以下のように計算される。

$$\begin{aligned}&N_{Tet}=\iiint_{Tet}\rho(S)dxdydz=\\&\iiint_{tet}\rho(s)\left|\det(\mathbf{J})\right|dt_1dt_2ds=\\&\iiint_{tet}M\,p(s)\left|\det\left(\mathbf{L}^{-1}\mathbf{A}^{-1}\right)\right|dt_1dt_2ds=\\&M\frac{1}{\left|a_1a_2a_3\right|}\frac{N_{in}}{N}\end{aligned} \tag{5.19}$$

ここで|det(**J**)|は式(A.2)による変数変換に対するヤコビアンである。**L** は直交行列なので、行列式の値は 1 になる。

5.2.4 実装

レイヤードサンプリングでは、*Tet''* に対する事前生成粒子群の包含関係が計算され、その内部に含まれる粒子数 N_{in} が得られる(*Insertion*)。そして生成粒子数 N_{tet} が式(5.12)から計算され(*Integration*)、N_{in} が N_{tet} を上回る場合に、*Tet''* 内部の粒子から N_{tet} 個の粒子が選ばれる(*Selection*)。選択された粒子に対して $\mathbf{L}^{-1}\mathbf{A}^{-1}$ を乗ずることで、オブジェクト座標が計算される(*Inverse Transformation*)。しかし、N_{in} が N_{tet} を上回らない場合には、*Tet''* 内部の全ての粒子に対するオブジェクト座標が計算された後、不足している $N_{tet}-N_{in}$ 個の粒子が棄却法によって生成される(*Rejection Sampling*)。ここで、$S_{\max}-S_{\min}$ で計算される、格子内部のスカラー値の範囲が小さくなる場合には、上述の処理が行われないことを留意しておく必要がある。

$S_{\max}-S_{\min}$ が 0 になる場合、式(5.12)において a_3 と N_{in} が 0 になる。これは、*Tet''* が平らに潰れたことを意味する。この場合、四面体格子のスカラー場は一定であるので、生成粒子数は一点積分で計算され、粒子は一様分布を形成する。

$S_{\max}-S_{\min}$ が微小な値δより小さくなる場合、*Tet''* の厚みが非常に薄くなり、N_{in} が非常に小さくなる。δはあらかじめ定義される、[0,1] の範囲の小数で、伝達関数の急峻な部分の幅よりも小さくなるように選ばれる。この場合、生成粒子数の計算にはモンテカルロ積分を、粒子の生成には棄却法を用いる。なぜならば、スカラー値の範囲が狭い分だけ四面体格子内部のスカラー値の変化は緩やかになるからである。

レイヤードサンプリングの実装を以下に示す。

if $S_{\max} - S_{\min} >= 0$

Calculate the transformation matrices **A** and **L** (*Transformation*)

Determine the inclusion of the pre-generated particles in *Tet''* and obtain N_{in} (*Insertion*)

Estimate N_{tet} by Equation 8 (*Integration*)

if $N_{tet} <= N_{in}$

Select N_{tet} particles included in *Tet''* (*Selection*)

Transform the selected particles into *Tet* by multiplying $\mathbf{L}^{-1}\mathbf{A}^{-1}$ (*Inverse Transformation*)

else

Select all particles included in *Tet''* (*Selection*)

Transform the selected particles into *Tet* by multiplying $\mathbf{L}^{-1}\mathbf{A}^{-1}$ (*Inverse Transformation*)

Generate N_{tet} - N_{in} particles by rejection sampling (*Rejection Sampling*)

else if $S_{\max} - S_{\min} = 0$

Estimate N_{tet} by single point integration

Generate particles by uniform sampling

else

Estimate N_{tet} by single point integration

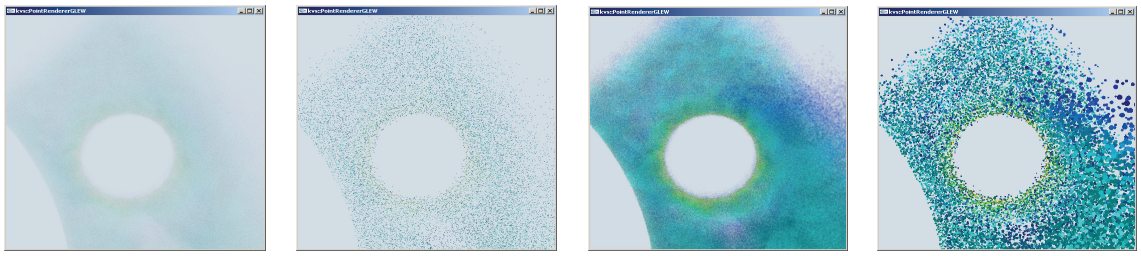
Generate particles by rejection sampling

上述した擬似コードにおいて、*Transformation*, *Insertion*, *Integration*, *Selection* そして *Inverse Transformation* はレイヤードサンプリングの主要な処理部分であり、以降ではこれらの処理をまとめて *Main* と呼ぶ。

5.3 実験と考察

5.3.1 粒子拡大手法の検証

粒子拡大手法の有効性を検証するために、12,936 の四面体格子からなるボリュームデータを用いた。実験環境は、1800MHz AMD Phenom X4 9150 Quad-core の CPU、4.0 GB の RAM、そしてグラフィックカートカードとして NVIDIA GeForce 9600GT GPU (ビデオメモリ 512MB)を搭載した計算機を使用した。粒子拡大手法実装以前の PBVR と、実装した PBVR のレンダリング速度と GPU で消費されたメモリを、幾つかのサブピクセルレベル、リピートレベルに対して計測した。表 5.1 に結果を示す結果から、両手法のレンダリング速度にほとんど差が見られないことがわかった。図 5.4 は両手法で生成された、ボリュームデータに対して拡大を行った画像を示す。以前の PBVR から得られた画像は薄く、粒子拡大手法を実装した PBVR から得られた画像は、期待されたとおり、拡大しても半透明感が変わらなかった。



(a) 粒子拡大無し
Sub-pixel level: 1
Repetition level: 100

(b) 粒子拡大無し
Sub-pixel level: 1
Repetition level: 1

(c) 粒子拡大有り
Sub-pixel level: 1
Repetition level: 100

(d) 粒子拡大有り
Sub-pixel level: 1
Repetition level: 1

図 5.4　粒子拡大手法による画像と粒子拡大を用いない画像の比較

表 5.1　粒子拡大手法の有無によるレンダリング速度の比較

Sub-pixel Level	Repetition Level	# of particles	GPU memory [M bytes]	[frame/sec] (Previous PBVR)	[frame/sec] (Opaque splatting)
1	100	4.2M	75.6	28.53	28.53
2	25	4.2M	83.0	28.62	28.99
3	25	9.4M	186.67	14.05	14.99
4	25	16.8M	331.83	9.09	9.04

粒子拡大手法実装以前の PBVR では、リピートレベルを増加させることで高画質な画像を得ることができた。従来手法との比較を行うため、同じデータに HAVS を適用した。図 5.5 にレンダリング結果を示す。リピートレベルと画質の関係を調べるために、格子数 222,414、頂点数 40,948 からなる“Blunt”データに粒子拡大手法を適用した。

この図から、粒子拡大手法による画像の画質が、リピートレベルを増加させることで改善される事がわかった。また HAVS による画像に、格子のソーティングエラーに起因するレンダリングアーティファクトが発生していることが確認できる。また、リピートレベルを 1 から 255 まで変化させて、レンダリング速度を計測した。表 5.2 に、生成粒子数とレンダリング速度を示す。生成粒子数がリピートレベルに比例して増加していることが、レンダリング速度がリピートレベルに反比例して減少していることが確認できる。HAVS のレンダリング速度は PBVR のリピートレベル 49 および 64 の場合と同じ結果であった。

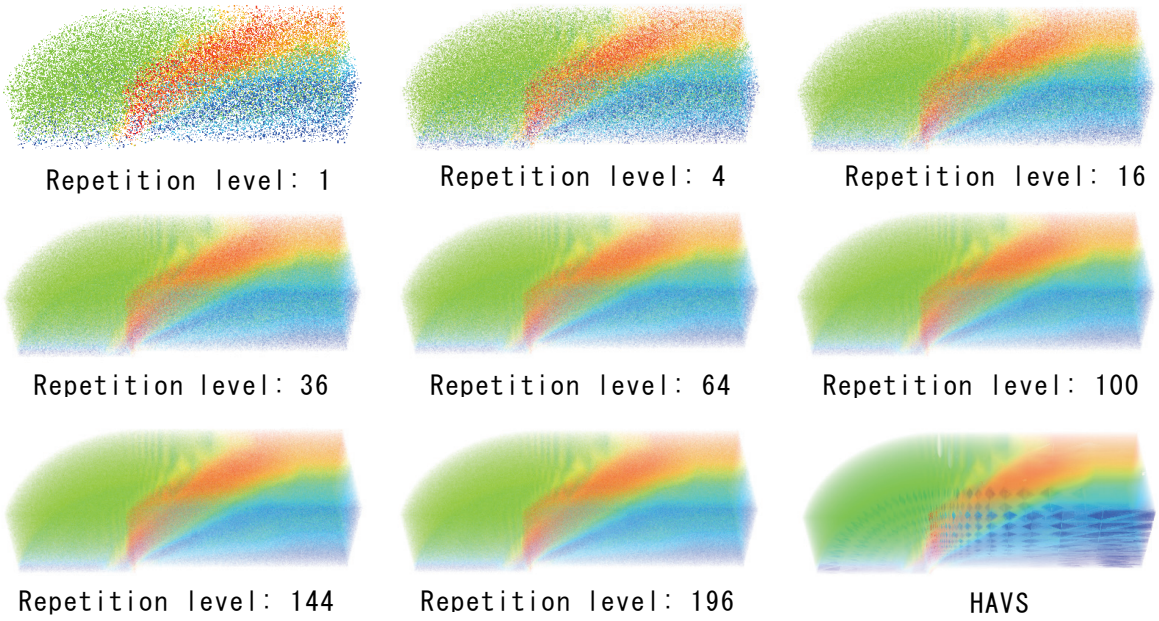


図 5.5　粒子拡大手法による画像と HAVS 画像の比較

表 5.2　粒子拡大手法と HAVS のレンダリング速度の比較

	PBVR															HAVS
Repetition level	1	4	9	16	25	36	49	64	81	100	121	144	169	196	225	
Number of particles [M]	0.05	0.2	0.5	0.8	1.3	1.8	2.5	3.2	4.1	5.0	6.1	7.3	8.5	9.9	11.0	
FPS	82	77	69	65	32	31	21	21	15	12	12	10	9	7	6	21

粒子拡大手法を用いて、大規模不規則格子ボリュームデータである Pump と Oral に適用した。リピートレベルを 144、画像解像度を 800x600 として、LOD 制御を用いた。図 5.6 にレンダリング画像を示す。左列が粗いレンダリング、右列が詳細なレンダリング画像である。また図 5.6(a,b-3,4)は拡大画像である。図 5.6（a）に Oral データに適用した結果を示す。レンダリング速度は、詳細なレンダリングにおいて 7.7[fps]、リピートレベル 1 を用いた荒いレンダリングにおいて 59.0[fps]であった。約 897 万の粒子が生成され、ボリュームデータ全体の境界面も同時に可視化した。Pump データに適用した結果を図 5.6（b）に示す。生成粒子数は約 1585 万であった。レンダリング速度は、詳細なレンダリングにおいて 5.3[fps]であり、リピートレベル 1 を用いた荒いレンダリングにおいて 59.0[fps]であった。荒いレンダリング時の拡大画像から、近くにある粒子の大きさが拡大され、拡大前と比較して半透明感が維持されていることが確認できる。

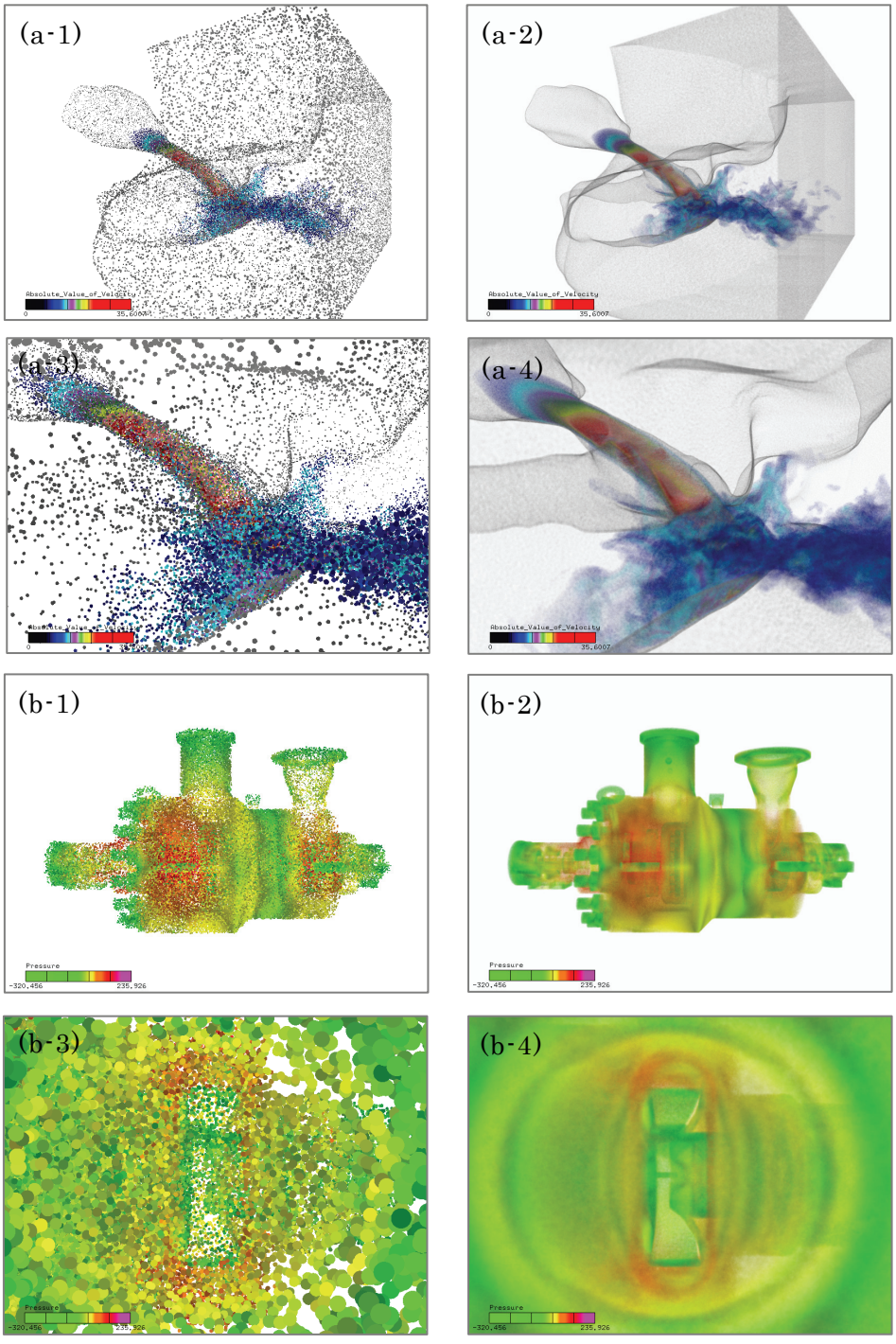


図 5.6　LOD を用いた粒子拡大手法によるレンダリング画像

5.3.2　レイヤードサンプリングの評価

PBVR の画質は粒子数推定の精度に密接に関連している。レイヤードサンプリ

ングによる体積積分の精度を評価するために、厳密解との比較を行った。ボリュームデータの体積積分の厳密解は、非常に簡単な場合にしか計算できないため、5つの四面体から構成された立方体の形状を持ち、内部のスカラー場が線形に分布するボリュームデータを作成し、実験を行った。図 5.7（a）はレンダリング結果と、適用された 4 つの急峻な峰を持つ伝達関数を示す。PBVR の従来の体積積分の手法である、重心位置における一点積分との比較も行った。この積分はモンテカルロ積分のサンプリング数を 1 点だけ用いた場合と同等である。実験では、レイヤードサンプリングとモンテカルロ積分による積分の値を計算し、積分に要した時間を計測した。モンテカルロ積分のサンプリング数は 1 点から 1000 万点まで

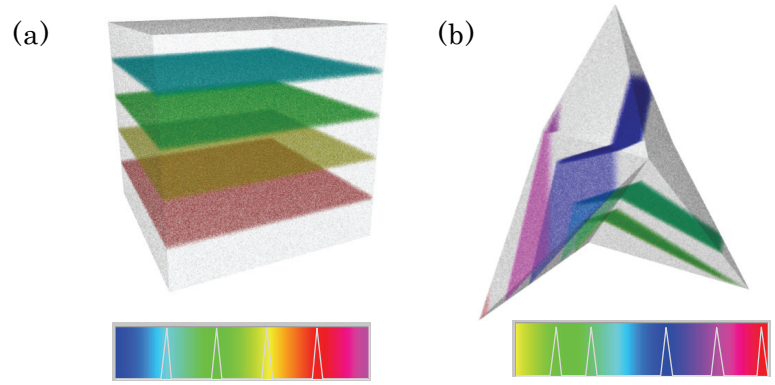


図 5.7　テストデータに対するレンダリング画像と伝達関数

表 5.3　立方体データに対する積分値の比較。厳密解=358,350

Monte Carlo integration			Layered sampling		
Num. of samples	Integ. value	Integ. time [msec]	Num. of pre-gen. parts.	Integ. value	Integ. time [msec]
1	0	0.003	10,000	358,350	3.3
10	0	7.4			
100	429,680	7.6			
1,000	374,609	9.3			
10,000	374,755	13.9			
100,000	359,906	44.8			
1,000,000	359,016	510.3			
10,000,000	357,846	4,939.6			

増加させ、レイヤードサンプリングの事前生成粒子数は 1 万点に設定した。レイヤードサンプリングの積分時間は、*Transformation*、*Insertion*、*Integration* の 3 つの処理に関して計測した。表 5.3 に示す結果から、モンテカルロ積分は最大で有効数字 2 桁の精度が得られたことがわかる。また、モンテカルロ積分のサンプリング数 10 以上のとき、レイヤードサンプリングはモンテカルロ積分よりも高速に、厳密解と等しい値を計算したことが確認できる。

表 5.4　事前生成に関する処理時間

Num. of pre-gen. parts. [million]	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0
Pre-generation [msec]	72	146	230	282	361	450	542	604
Main [msec]	85	168	224	273	311	328	464	508
Rejection Sampling [msec]	9727	5700	2705	1092	0	0	0	0
Total [msec]	9884	6014	3159	1920	672	778	1006	1112

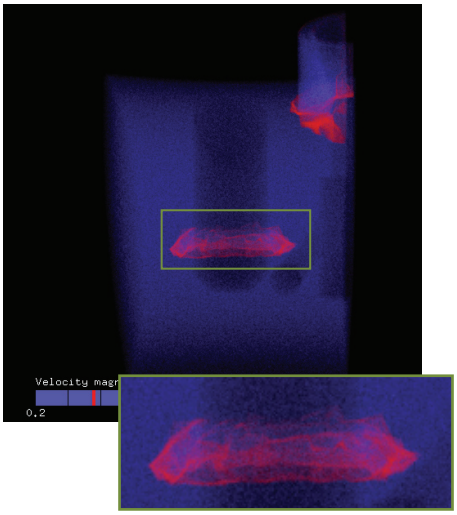
レイヤードサンプリングの処理時間は、画質とは異なり、事前生成粒子数 N に影響されると考えられる。このことを検証するため、N の値を変化させて、粒子の事前生成、*Main* そして *Rejection Sampling* の処理時間をそれぞれ計測した。実験には、図 5.7(b)に示す、4 つの四面体格子から構成されるデータと 5 つの急峻な峰を持つ伝達関数を用いた。表 5.4 から、合計時間が単純に N の増加に比例していないことがわかる。これは N が生成粒子数 N_{tet} を超えない場合、*Rejection Sampling* が実行されていることに起因する。この実験では N が 200 万以下で *Rejection Sampling* が実行されていることが確認できる。レイヤードサンプリングと比べて大きな処理時間を必要としている *Rejection Sampling* は、十分な数の事前生成粒子数 N があれば実行されない。十分な N を計算するためには、ボリュームデータの全四面体格子に対して粒子数推定を行い、最大値を求める必要があると考えられる。その最大値を利用して、必要十分な N を計算することができるだろう。しかし前もって全四面体格子の処理を行うことは、無視できない計算時間になると考えられる。

レイヤードサンプリングの有効性を検証するため、実データ、SPX と Aorta への適用を行った。従来の粒子生成手法も適用し、画質と計算速度、レンダリング速度を比較した。実験には、厚みを持った等値面状の領域を形成するように、急峻な峰を一つ持った伝達関数を用いた。図 5.8 は SPX の、図 5.9 は Aorta のレン

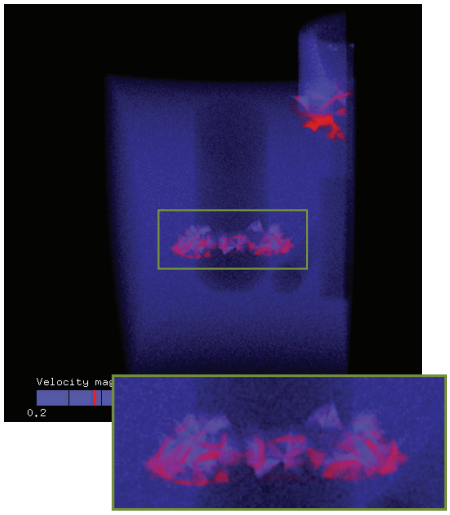
ダリング結果であり、それぞれの図中において（a）はレイヤードサンプリングにより、（b）は以前の手法により生成された画像である。また（c）は適用した伝達関数である。図 5.8（b）から、赤色の部分に関して、不十分なサンプリングによる不連続的な模様が見られる。一方図 5.8（a）から、赤色の曲面が可視化されていることが確認できる。図 5.9（b）では、白の曲面に穴が開いているように見えるが、（a）ではそれが改善されている。表 5.5 に、事前生成時間、粒子生成時間、発生粒子数、レンダリング速度を示す。レイヤードサンプリングの粒子生成時間が従来手法を上回っていることが確認できる。これは従来手法で用いられている一点積分が非常に高速なことに起因すると考えられる。表 5.5 から、SPX の生成粒子数はレイヤードサンプリングの方が従来手法よりも多いのに対して、Aorta は従来手法の方がレイヤードサンプリングよりも多くなっている事がわかる。一点積分は積分計算に用いるサンプリング点が等値面領域に入らなかった場合、粒子数を過小に計算する。SPX に対する実験では、この効果が多く発生したと考えられる。また、サンプリング点がたまたま等値面領域に含まれた場合、逆に粒子数を過大に計算してしまう。Aorta に対する実験では、この効果が多く発生したと考えられる。図 5.9 の（a）と（b）を比べると、白い等値面領域に関して、従来手法の方がレイヤードサンプリングよりも濃く見える部分がある。これはある部分で粒子数が過大評価されたことに起因すると考えられる。

表 5.5　事前生成粒子数 100,000 を用いた場合の処理時間とレンダリング速度

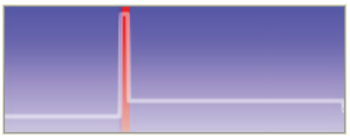
	SPX		Aorta	
	T1	T2	T1	T2
Pre-gen. time [msec]	703		750	
Sampling time [sec]	5.6	1.1	290.7	4.7
Num. particles [M]	5.16	5.12	17.74	17.76
FPS	18.6	18.8	7.6	7.6



(a) レイヤードサンプリング

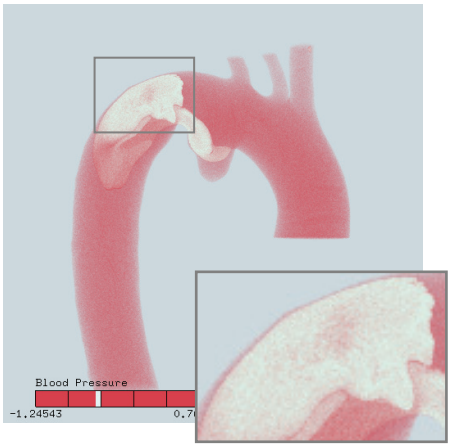


(b) メトロポリスサンプリング

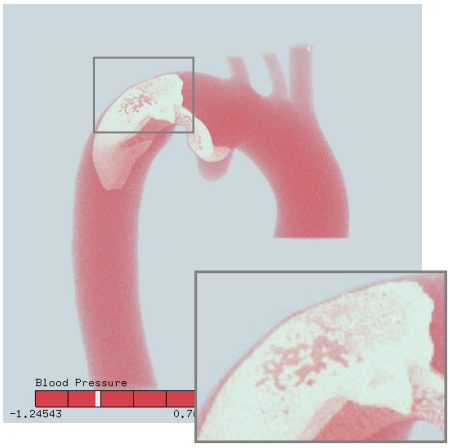


(c) 伝達関数

図 5.8　SPX に対するレンダリング画像と伝達関数



(a) レイヤードサンプリング



(b) メトロポリスサンプリング

(c) 伝達関数

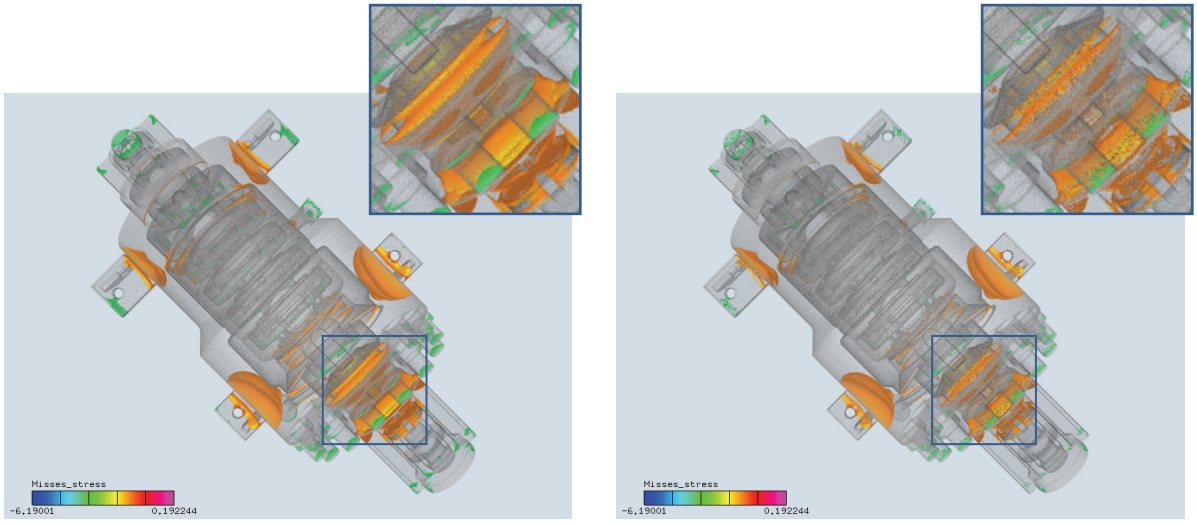
図 5.9　Aorta に対するレンダリング画像と伝達関数

5.3.3 大規模データへの適用例

レイヤードサンプリングを大規模データに適用し、従来手法との比較を行った。実験には、Pump データを構成する四面体二次要素を 8 個の四面体格子に分割した、格子数 210,318,160 のボリュームデータを用いた。この実験のために、クロック周波数 2.83GHz を持つ Intel Core2 Quad の CPU と 8.0GB の RAM、そしてグラフィックカートカードとして、1.5GB のビデオメモリを持つ NVIDIA GeForce GPX280 GPU を搭載した計算機を用いた。そして 4 つのスレッドを用いて並列に粒子生成を行った。各スレッドがそれぞれ、4i, 4i+1, 4i+2, 4i+3 番目の格子（i = 0, 1,…, 52,579,539）を処理した。図 5.10（a）にレイヤードサンプリングの、図 5.10（b）に従来手法のレンダリング結果を示す。図 5.10（c）には適用した伝達関数を示す。この伝達関数は 5 つの急峻な峰を持つ。ボリュームデータの形状を明確にするため、その境界面も同時に可視化した。境界面の形状をポリゴンデータとして抽出し、その上に 700 万の粒子を一様分布させた。図 5.10（a）と（b）を比較すると、従来手法に見られる等値面領域に無数の穴が、レイヤードサンプリングでは明らかに改善されていることが確認できる。表 5.6 に事前生成時間、粒子生成時間、生成粒子数、レンダリング速度を示す。生成粒子数は、境界面上の粒子数を含まない。表 5.6 から、レイヤードサンプリングによる粒子生成時間が従来手法に比べて非常に巨大になっていることがわかる。従来手法のレンダリング速度がレイヤードサンプリングのレンダリング速度よりも高速であるが、これはレイヤードサンプリングによる生成粒子数が従来手法のそれよりも大きいからだと考えられる。

表 5.6　Pump に対する粒子生成時間とレンダリング速度

	Layered sampling	Metropolis sampling
Pre-gen. time [msec]	3.9	
Num. particles [M]	9.8	5.2
Sampling time [sec]	1163.9	27.7
FPS	7.9	10.2



(a) レイヤードサンプリング　　(b) メトロポリスサンプリング

(c) 伝達関数

図 5.10　Pump に対するレンダリング画像と伝達関数

5.4　まとめ

従来の PBVR はボリュームデータの拡大時に輝度値が減少し、疎な画像を生成した。この問題を解決するために、視点からの距離に応じて粒子半径を変える手法を提案した。幾つかのボリュームデータに適用し、拡大時にも画像の不透明度が維持されることを確認した。

急峻な伝達関数に適用可能な、不規則四面体格子ボリュームデータ向けの粒子生成手法である、四面体向けレイヤードサンプリングを提案した。レイヤードサンプリングでは、区分線形伝達関数から区分線形密度関数を作成し、それを参照してレギュラーボックス内部に粒子を事前生成する。体積積分と粒子生成は事前生成粒子群を用いて行われる。実験により、粒子数を計算するための体積積分の精度と画質、粒子生成時間について検証した。積分の精度について、単純な形状と線形に分布するスカラー場について厳密解を計算し、モンテカルロ積分（従来手法はモンテカルロ積分の特別な場合）と比較した。その結果、モンテカルロよりも高精度な結果が得られた。実データに適用し画質と粒子生成時間を比較した。

従来手法に顕著に見られた等値面領域に関するアーティファクトを、レイヤードサンプリングで改善することができた。しかしレイヤードサンプリングの粒子生成は従来手法よりも遅く、ボリュームデータの格子数が多いほどその差が大きくなった。

粒子生成時間を高速化するため、従来手法とのハイブリッド化を計画している。粒子の密度分布が緩やかな部分では、従来手法でも十分な画質が得られると考えられる。そのため伝達関数の形状と格子のスカラー場の変化率から、レイヤードサンプリングか従来手法か、どちらか適した手法を適用して粒子生成を行うシステムの開発が望まれる。また、粒子生成時間は事前生成粒子数に依存する。ハイブリッド化したシステムに適した形で、最適な事前生成粒子数を計算する方法の開発が必要とされる。

第6章　六面体向けレイヤードサンプリング

有限要素法では不規則四面体格子同様に、不規則六面体格子も頻繁に用いられ、そのボリュームレンダリング技術の開発は重要である。この章では、急峻な伝達関数が与えられても高品位な粒子生成を行うことができる、六面体向けレイヤードサンプリングを提案する。六面体向けレイヤードサンプリングは、区分線系伝達関数を構成する区分毎に粒子生成を行う手法であり、体積積分、変数変換、ギブス法、逆関数法の四つの処理から構成されている。六面体の頂点に定義されたスカラー値は、局所座標$[0,1]^3$ 内部で三重線形補間され、有限要素法で用いられる形状関数によって六面体内部に写像される。この複雑な形状のスカラー場に沿って確率的に粒子を生成するために、確率変数に対して補間関数を利用した変数変換を行う。そして変数変換の結果生じたヤコビアンに従って確率変数を生成するために、マルコフ連鎖モンテカルロ法の一種であるギブス法を利用する。また、ギブス法で生成する必要のある一変数の確率密度関数に従って確率変数を生成するために、逆関数法を利用する。提案手法では、逆関数法に出表する特異点を持つ関数に対して近似関数を与えて計算を行う。この近似関数は近似の度合いを表すパラメータを持つ。また、従来の PBVR では生成粒子数を計算するために、格子中心における一点積分を行っていたが、本手法ではより高精度に計算を行うために、ガウスールジャンドル積分を適用した。提案手法のガウスールジャンドル積分は、積分点の増加と格子の分割と併用している。提案手法で粒子生成を行った後に、従来の PBVR の枠組みでレンダリングを行う。

PBVR では、粒子密度ρは不透明度αから計算される。六面体向けレイヤードサンプリングでは、スカラー値 s に対する区分線形密度関数$\rho(s)$を用いて計算を行う。この関数は粒子密度推定関数$\rho(a)$と区分線形伝達関数の不透明度$\alpha(s)$から計算される。区分線形密度関数を構成する節点間の値は線形補間により計算される。

6.1　生成粒子数の計算

各セルで密度分布ρを体積積分することによって、生成粒子数 N を計算する。大域座標 **X** に関する体積積分は、以下のように局所座標 **x** に関する積分へと変形される。

$$N = \int_{\mathrm{Hex}} \rho(s(\mathbf{x}))d\mathbf{X} = \int_{[0,1]^3} \rho(s(\mathbf{x}))\left|\det(\mathbf{J}_s(\mathbf{x}))\right| d\mathbf{x} \tag{6.1}$$

ここで $\mathbf{J}_s$ は、形状関数のヤコビ行列関数、Hex は着目セルの領域を意味する。

六面体レイヤードサンプリングは上記の積分を計算するために、ガウスールジャンドル積分を用いる。この手法は、特定の分点の値における関数の値を、重み付きで総和することによって、積分値を数値的に計算することができる。重みと分点の座標は事前に計算されており、数表として与えられている。この手法は、n 個の分点が与えられたとき、$2n-1$ 次の多項式に関する積分が計算される。

本論文では、ガウスールジャンドル積分の精度に関する知見を得るため、局所座標において格子の分割を行い、分割された格子毎にガウスールジャンドル積分を適用した。$[0,1]^3$ の領域を均等に $resol^3$ 分割し、分割された格子のインデックスを f、g、h $(0 \le f,g,h \le resol-1)$とする。分割された領域の一辺の長さは $l=1/resol$ となる。重みを w とすると、ある分割された格子の生成粒子数 $N_{f,g,h}$ は以下のように計算される。

$$N_{f,g,h} = \frac{l^3}{8}\sum_i^n\sum_j^n\sum_k^n w_i w_j w_k \rho(s)\left|\det(\mathbf{J}_s)\right| \tag{6.2}$$

重み w_i に対応した分点の座標をξ_i とすると、局所座標の x 軸に対する座標変換は以下のようになる。

$$x = \frac{l}{2}(\xi_i + 1) + l \cdot f \tag{6.3}$$

y 軸、z 軸についても同様である。

6.2　媒介変数表示

粒子密度がスカラー値 s_1 と s_2 の間で 0 でない値を持つとき、粒子生成を行う

領域は、s_1を閾値とする等値面とs_2を閾値とする等値面に挟まれた閉領域となる。しかしながら以前の手法は、サンプリング点をその閉領域の外側も含めたセル全体にばらまいていたため、サンプリングの失敗が頻出していた。区分線形密度関数を構成する区分毎の粒子生成は、そのようなサンプリングの失敗を減少させる。なぜなら、サンプリング実施をその閉領域内部に限定するからである。これを実表するために、粒子生成に用いる変数を局所座標(x,y,z)からスカラー値sを陽に含む変数に変換し、閉領域内部に粒子を直接生成する。局所座標(x,y,z)内部で三重線形補間されたスカラー場が次式で書かれるとする。

$$s = Axyz + B_0xy + B_1yz + B_2zx + C_0x + C_1y + C_2z + D \tag{6.4}$$

ここでAからDは三重線形補間式の係数である。六面体向けレイヤードサンプリングは、x、y、z軸、それぞれに沿った3つの変数変換を用いる。変数変換の式は式(6.4)から導かれ、s、t、uを新しい変数とする。以下で、x、y、z座標に関する変換式を示す。

$$\begin{gathered} x = \frac{s-(B_1tu + C_1t + C_2u + D)}{Atu + B_0t + B_2u + C_0} \\ y = t \\ z = u \end{gathered} \tag{6.5}$$

$$\begin{gathered} x = u \\ y = \frac{s-(B_2tu + C_2t + C_0u + D)}{Atu + B_1t + B_0u + C_1} \\ z = t \end{gathered} \tag{6.6}$$

$$\begin{gathered} x = t \\ y = u \\ z = \frac{s-(B_0tu + C_0t + C_1u + D)}{Atu + B_2t + B_1u + C_2} \end{gathered} \tag{6.7}$$

提案手法ではこれらの変数を確率変数と捉え、統計的に値を生成する。

6.3　ギブス法による粒子生成

局所座標中の一様分布を式(6.5)-(6.7)で変換すると、以下のように表される。

$$\iiint 1\,dx\,dy\,dz = \iiint \left|\det\left(\mathbf{J}(s,t,u)\right)\right| ds\,dt\,du \tag{6.8}$$

ここで$\mathbf{J}$は式(6.5)-(6.7)の変数変換のヤコビアンである。以降、$|\det \mathbf{J}|$を$V(s,t,u)$と書く。

式(6.5)-(6.7)の変換に対応した$V(s,t,u)$をそれぞれV_x、V_y、V_zと表記し、それら

は以下のように表される。

$$V_x(s,t,u) = 1/\left|Atu + B_0 t + B_2 u + C_0\right| \tag{6.9}$$

$$V_y(s,t,u) = 1/\left|Atu + B_1 t + B_0 u + C_1\right| \tag{6.10}$$

$$V_z(s,t,u) = 1/\left|Atu + B_2 t + B_1 u + C_2\right| \tag{6.11}$$

$V(s,t,u)$は s に依存しないため、この関数を $V(t,u)$とも書く。

提案手法は $V(t,u)$に従って t と u を生成するために、マルコフ連鎖モンテカルロ法の単純な場合であるギブス法を用いる。この手法は、まず $V(t,u)$の t を定数 t_i-1 として、$V(t_{i-1}, u)$を確率分布として u_i をサンプリングし、次は $V(t,u)$の u を定数 u_i とし、$V(t, u_i)$を確率分布として t_i を生成する。これを繰り返すことで、もとの確率分布から点列をサンプリングする手法である。

また同時に、s を密度関数に従うように棄却法を用いて生成する。六面体向けレイヤードサンプリングは、x、y、z 軸に対応した変数変換の式(6.5)-(6.7)を逐次取り替えながら計算を行う。

提案手法は各セルの内部で、区分線系密度関数の区分毎に粒子生成を行う。初めに生成粒子数を計算し、次に変数変換を求める。そしてギブス法と逆関数法を用いて粒子を生成する。以下に提案手法のアルゴリズムを示す。

アルゴリズム概要 ： GibbsSampling

```
14:  VolumeData volume
15:  HexahedralCell cell
16:  PiecewiseLinearDensityFunction df
17:  for each cell in volume
18:    for each interval in df
19:      N = CalculateNumOfParticles(cell,df)
20:      axis = 0
21:      i = 1
22:      t_{i-1} = Random()
23:      while i < N
24:        if axis = 3 then
25:          axis = 0
26:        end if
27:        SelectEquations( axis )
28:        Generate u_i from V_axis(t_{i-1},u)
29:        Generate t_i from V_axis(t,u_i)
30:        Generate s_i from ρ(s)
31:        Transform s_i,t_i,u_i to x,y,z
32:        t_{i-1}=t_i
33:        if 0 <= x,y,z <=1 then
```

```
34:          Transform x,y,z to global coord.
35:          i++
36:        end if
37:        axis++
38:      end while
39:    end for
40:  end for
```

上述のアルゴリズムにおいて、axis の値 0、1、2 はそれぞれ x、y、z 軸を表す。$V_{axis}(t,u)$は式(6.9)-(6.11)で定義される確率密度分布を意味する。先述した通り、ギブス法を用いて各変数に定数を代入しながら確率変数を生成する。提案手法では、定数が代入された一変数関数 $V(t_{i-1}, u)$もしくは $V(t, u_i)$に対して確率変数を生成するために、逆関数法を用いる。詳細は次節に記述する。関数 SelectEquations(axis)は、入力された axis の値に対応した式(6.5)-(6.7)及び式(6.9)-(6.11)を選択する。

6.4　逆関数法

ギブス法は $V(t,u)$の t、もしくは u のどちらかを定数として計算を進める。このとき $V(t,u)$は以下のような 1 変数の関数として表される。

$$V(p)=\frac{1}{|ap+b|}\quad (p=t,u\in[0,1]) \tag{6.12}$$

ここで a、b は式(6.9)-(6.11)を変数 p について整理したときの係数である。式(6.12)の不定積分は対数関数として計算することができるため、確率変数 p は逆関数法を用いて計算することができる。この手法は、q を一様分布に従う乱数とし、$V(p)$の累積関数の逆関数である関数 f（$p=f(q)$となる）を計算する必要がある。そして q を関数 f で変換することによって V に従う p が得られる。しかし、式(6.12)から、$0<-b/a<1$ となる場合、$p=-b/a$ で積分の値が発散し、明らかに累積関数を計算することが不可能だと解る。この問題を解決するために、本手法では式(6.12)の特異点 $p=-b/a$ の近傍において、元の関数の平方根に比例する関数で近似した。以下にこの近似式を導入した確率密度分布関数を示す。

$$V(p)=\begin{cases} \dfrac{1}{|ap+b|} & \begin{pmatrix} 0 \le p < P_{-}\ or \\ P_{+} \le p \le 1 \end{pmatrix} \\ \dfrac{1}{\sqrt{\zeta|ap+b|}} & \left(P_{-} \le p < P_{+}\right) \end{cases} \tag{6.13}$$

ここで $P_{+}=-b/a+\zeta/|a|, P_{-}=-b/a-\zeta/|a|$ である。また、常に $P_{-}<-b/a<P_{+}$ が成り立つ。ζ はユーザー指定の正数である。

図 6.1 に式(6.13)のグラフを示す。式(6.13)の関数が $P_{+,-}$で値が連続になるように関数を構成している。

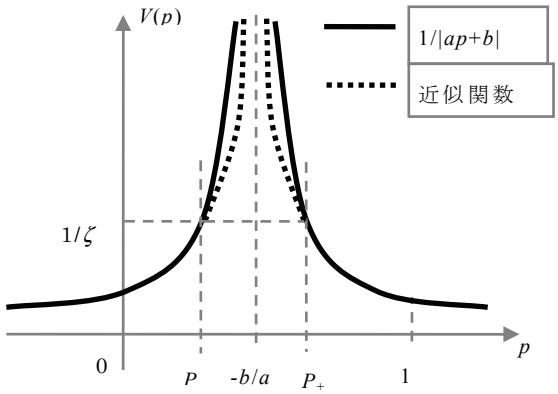


図 6.1　逆関数法で用いる近似関数

逆関数法は元の関数の累積関数の逆関数を用いて計算を行う手法である。式(6.13)で近似関数の値は式(12)と同様に、$p=-b/a$ で無限大に発散するが、主値積分を行うことで累積関数 f^{-1} の値は有限に収束する。そのため、関数 $V(p)$が無限大に発散する値を持っていても、逆関数法の計算を実行することができる。

累積関数の計算について、変数 p の値の範囲について場合分けする必要がある。図 1 から分かるとおり、この関数 $V(p)$は p について、$-b/a$ と P_{+}と P_{-}で分けられる四つの区間 $p \le P_{-}$、$P_{-}<p \le -b/a$、$-b/a<p \le P_{+}$、$P_{+}<p$から構成される。これらの区間に対して、変数 p の取りうる範囲[0,1]は、以下に示す 10 通りの場合に分けられる。

$$
\begin{aligned}
&\text{case}\,0:\ 1 \le P_- \\
&\text{case}\,1:\ 0 \le P_-,\ P_- < 1 \le -b/a \\
&\text{case}\,2:\ 0 \le P_-,\ -b/a < 1 \le P_+ \\
&\text{case}\,3:\ 0 \le P_-,\ P_+ < 1 \\
&\text{case}\,4:\ P_- < 0 \le -b/a,\ P_- < 1 \le -b/a \\
&\text{case}\,5:\ P_- < 0 \le -b/a,\ -b/a < 1 \le P_+ \\
&\text{case}\,6:\ P_- < 0 \le -b/a,\ P_+ < 1 \\
&\text{case}\,7:\ -b/a < 0 \le P_+,\ -b/a < 1 \le P_+ \\
&\text{case}\,8:\ -b/a < 0 \le P_+,\ P_+ < 1 \\
&\text{case}\,9:\ P_+ < 0
\end{aligned}
$$

それぞれの場合に対して、$V(p)$の累積関数 f_{ij}^{-1} $(i,j=0,1,2,3)$は以下のように計算される。

$$
\begin{aligned}
f_{00}^{-1}(p) &= f_0^{-1}(p,0) && (\text{case}\,0) \\
f_{10}^{-1}(p) &= f_1^{-1}(p,P_-) + f_{00}^{-1}(P_-) && (\text{case}\,1) \\
f_{20}^{-1}(p) &= f_2^{-1}(p,-b/a) + f_{10}^{-1}(-b/a) && (\text{case}\,2) \\
f_{30}^{-1}(p) &= f_3^{-1}(p,P_+) + f_{20}^{-1}(P_+) && (\text{case}\,3) \\
f_{11}^{-1}(p) &= f_1^{-1}(p,0) && (\text{case}\,4) \\
f_{21}^{-1}(p) &= f_2^{-1}(p,-b/a) + f_{11}^{-1}(-b/a) && (\text{case}\,5) \\
f_{31}^{-1}(p) &= f_3^{-1}(p,P_+) + f_{21}^{-1}(P_+) && (\text{case}\,6) \\
f_{22}^{-1}(p) &= f_2^{-1}(p,0) && (\text{case}\,7) \\
f_{32}^{-1}(p) &= f_3^{-1}(p,P_+) + f_{22}^{-1}(P_+) && (\text{case}\,8) \\
f_{33}^{-1}(p) &= f_3^{-1}(p,0) && (\text{case}\,9)
\end{aligned}
\qquad (6.14)
$$

但し、関数 f_i^{-1} $(i=0,1,2,3)$は次のように定義される。

$$
f_i^{-1}(p,p_0) = {}_pf_i^{-1}(p) - {}_pf_i^{-1}(p_0) \quad i=0,1,2,3
$$

$$
{}_pf_0^{-1}(p) = \frac{1}{-|a|}\log|ap+b|
$$

$$
{}_pf_1^{-1}(p) = \frac{2}{-|a|\sqrt{\zeta}}\sqrt{|ap+b|}
$$

$$
{}_pf_2^{-1}(p) = \frac{2}{|a|\sqrt{\zeta}}\sqrt{|ap+b|}
$$

$$
{}_pf_3^{-1}(p) = \frac{1}{|a|}\log|ap+b|
$$

式(6.14)の逆関数は以下のように計算される。

$$
\begin{aligned}
f_{00}(q) &= f_0(q, C_{00}) \quad (\text{case } 0)\\
f_{10}(q) &= f_1(q, C_{10}) \quad (\text{case } 1)\\
f_{20}(q) &= f_2(q, C_{20}) \quad (\text{case } 2)\\
f_{30}(q) &= f_3(q, C_{30}) \quad (\text{case } 3)\\
f_{11}(q) &= f_1(q, C_{11}) \quad (\text{case } 4)\\
f_{21}(q) &= f_2(q, C_{21}) \quad (\text{case } 5)\\
f_{31}(q) &= f_3(q, C_{31}) \quad (\text{case } 6)\\
f_{22}(q) &= f_2(q, C_{22}) \quad (\text{case } 7)\\
f_{32}(q) &= f_3(q, C_{32}) \quad (\text{case } 8)\\
f_{33}(q) &= f_3(q, C_{33}) \quad (\text{case } 9)
\end{aligned}
\tag{6.15}
$$

但し、f_i と C_{ij} $(i,j=0,1,2,3)$ は次のように定義される。

$$
\begin{aligned}
f_0(q, C) &= \frac{1}{-|a|} e^{-|a|(q-C)} - \frac{b}{a}\\
f_1(q, C) &= \frac{-|a|\zeta}{4}(q-C)^2 - \frac{b}{a}\\
f_2(q, C) &= \frac{|a|\zeta}{4}(q-C)^2 - \frac{b}{a}\\
f_3(q, C) &= \frac{1}{|a|} e^{|a|(q-C)} - \frac{b}{a}
\end{aligned}
$$

$$
\begin{aligned}
C_{00} &= -{}_pf_0^{-1}(0)\\
C_{10} &= -{}_pf_1^{-1}(P_-) + {}_pf_0^{-1}(P_-) + C_{00}\\
C_{20} &= -{}_pf_2^{-1}(-b/a) + {}_pf_1^{-1}(-b/a) + C_{10}\\
C_{30} &= -{}_pf_3^{-1}(P_+) + {}_pf_2^{-1}(P_+) + C_{20}\\
C_{11} &= -{}_pf_1^{-1}(0)\\
C_{21} &= -{}_pf_2^{-1}(-b/a) + {}_pf_1^{-1}(-b/a) + C_{11}\\
C_{31} &= -{}_pf_3^{-1}(P_+) + {}_pf_2^{-1}(P_+) + C_{21}\\
C_{22} &= -{}_pf_2^{-1}(0)\\
C_{32} &= -{}_pf_3^{-1}(P_+) + {}_pf_2^{-1}(P_+) + C_{22}\\
C_{33} &= -{}_pf_3^{-1}(0)
\end{aligned}
$$

図 6.2 は関数対数関数である f_2 と 2 次関数である f_3 のグラフの一部を示す。実線が対数関数、破線が 2 次関数である。$q=P_+$ で対数関数と 2 次関数が接続している。

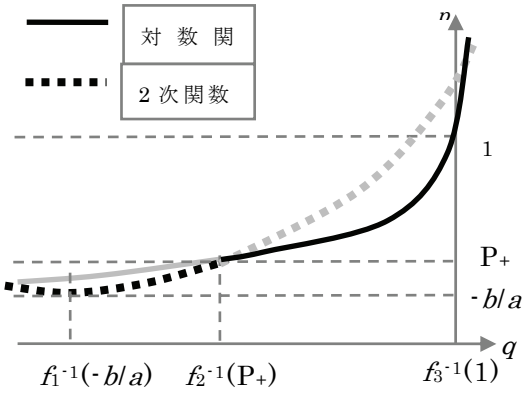


図 6.2　逆関数法で用いる累積関数の逆関数

6.5　実験結果と考察

提案手法は、生成粒子数の計算に関して分点数と格子分割数を、粒子生成に関して近似関数の幅ζをパラメータに持つ。積分精度や画質、計算速度に対するこれらのパラメータの影響を検証するための実験を行い、実データに対する適用を行った。実験環境は、Quad-Core AMD Opteron 2350 CPU、クロック周波数 2000 MHz、32GByte RAM を搭載している PC を用いた。GPU は、1GByte VRAM を搭載している NVIDIA Quadro FX 4700 を用いた。

6.5.1　数値積分の精度

ガウスールジャンドル積分は n 個の分点を適切に選ぶことで、$2n$-1 次の多項式による近似で計算を行うが、伝達関数はユーザー指定により任意の形状をとりうるため、提案手法の積分精度と計算速度に関する検証が必要である。セルの分割数とガウスールジャンドル積分の分点数を変化させて積分値を計算し、その計算速度を計測した。そして積分値の指標とするために、モンテカルロ積分による計算を行い、積分値と計算速度を比較した。実験には、セル数 64、頂点数 125 からなる六面体のテストデータと、不透明度にピークを一つだけ持つ伝達関数を用いた。

表 6.1 に提案手法の数値積分の値を、表 2 に数値積分の実行時間を示す。両表において、行がセルの分割数、列がガウスールジャンドル積分の分点数を意味する。従来の PBVR は、セルの重心位置の密度を用いた一点積分であり、分点数 1、分割数 1 の場合と等しい。セル内部の総サンプリング数は、分点数と分割数の積から計算される。表 6.3 にモンテカルロ積分による積分値と実行時間を示す。モンテカルロ積分の誤差は、$\sqrt{\log\log N / N}$ なので、サンプリング数が 10^4 の場合概ね有効数字 2 桁程度の、10^6 の場合有効数字 3 桁程度の、10^8 の場合有効数字 4 桁程度の信頼性があると考えられる。表 6.1 の結果から、計算精度は分点数と分割数の積に関係し、その値が大きいほど精度が高いという知見が得られる。分割数と分点数の積が等しい場合は、サンプリング数も等しくなるが、サンプリング点の位置が異なる。表 6.2 から、サンプリング数が等しい時は計算時間も概ね等しくなっていることが分かる。また、表 6.1－6.3 から、分割ガウスールジャンドル法はモンテカルロ積分と同程度の精度の計算を行う場合、モンテカルロ積分よりも高速であることが解る。有効数字 3 桁程度の精度の場合、分割ガウスールジャンドル法は 200[msec]以内に計算が終了しているのに対し、モンテカルロ積分は 4500[msec]以上の時間を要している。それ以上の精度の場合、その差は更に大きくなっていることが確認できる。

表 6.1　分割ガウスールジャンドル積分による積分値[K]。モンテカルロ積分から推定される積分値は約 1、556、000（有効数字 4 桁）である。有効数字 2 桁まで一致した値は緑字で、3 桁は青字で、4 桁まで一致した値は赤字で示してある。

		分点数						
		1^3	2^3	4^3	6^3	8^3	10^3	20^3
分割数	1^3	0	1,748	2,571	1,479	1,432	1,360	1,562
	2^3	88	1,165	1,808	1,648	1,529	1,541	1,556
	3^3	187	1,345	1,599	1,551	1,558	1,557	1,556
	4^3	299	1,602	1,597	1,559	1,556	1,556	1,556
	5^3	136	1,477	1,563	1,556	1,556	1,556	1,556
	6^3	195	1,602	1,552	1,556	1,556	1,556	1,556
	7^3	223	1,589	1,558	1,556	1,556	1,556	1,556
	8^3	177	1,516	1,557	1,556	1,556	1,556	1,556
	9^3	195	1,575	1,557	1,556	1,556	1,556	1,556
	10^3	206	1,556	1,556	1,556	1,556	1,556	1,556

表 6.2　分割ガウスールジャンドル積分の計算時間。単位は[msec]。

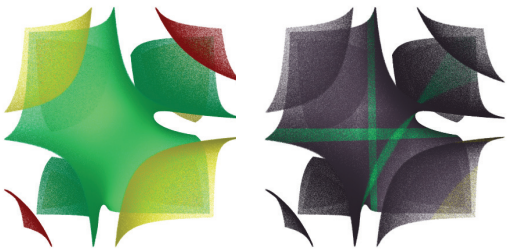
		分点数						
		1^3	2^3	4^3	6^3	8^3	10^3	20^3
分割数	1^3	0	0	0	1	2	4	32
	2^3	0	0	2	7	16	32	255
	3^3	0	1	7	23	55	108	860
	4^3	0	2	17	55	130	254	2,025
	5^3	1	4	32	108	254	495	3,955
	6^3	1	7	56	185	439	856	6,833
	7^3	2	12	89	295	696	1,358	10,891
	8^3	3	17	133	442	1,044	2,037	16,265
	9^3	4	25	189	629	1,487	2,901	23,158
	10^3	6	34	259	863	2,039	3,978	31,765

表 6.3　モンテカルロ積分による
サンプル数、積分値、計算時間。

サンプリング数	積分値	計算時間[msec]
1,000	1,673,475	5
10,000	1,534,096	45
100,000	1,569,760	457
1,000,000	1,554,026	4,529
10,000,000	1,556,583	44,944
100,000,000	1,556,007	449,540

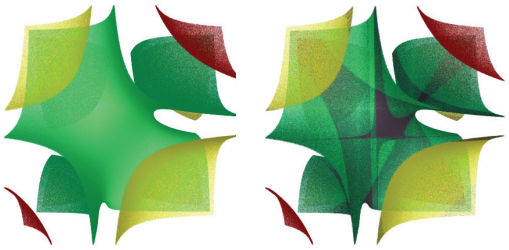
6.5.2　逆関数法の近似パラメータに関する検証

提案手法を自作したセル一つから構成される 2 種類のテストデータに適用し、ユーザー指定のパラメータζを変化させて、画質と計算速度の検証を行う。ζの値が大きいほど式(6.13)に対して近似関数の占める割合が大きくなり、ゼロに近づくほど小さくなる。図 6.3、6.4 に結果を示す。左列が適用結果であり、右列が近似関数によって計算された粒子を黒で塗った結果である。図 6.3 において、パラメータの値に関わらず画質に変化がないことが確認される。また、パラメータの値が大きいほど高速であることが確認される。また、左右の絵を比較しても、近似関数の使用に起因する不連続性は見られないことが解る。図 6.4 の結果からは、パラメータの値が大きい時にアーティファクトが観察される。右列の絵から、アーティファクトは近似関数によって生じていることが確認される。

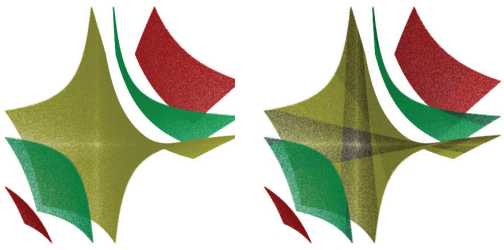


ζ=1, sampling time 21 [sec]

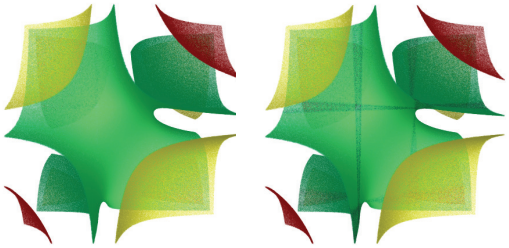
ζ=1, sampling time 15 [sec]



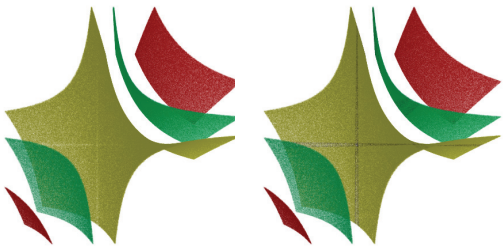
ζ=0.1, sampling time 61 [sec]



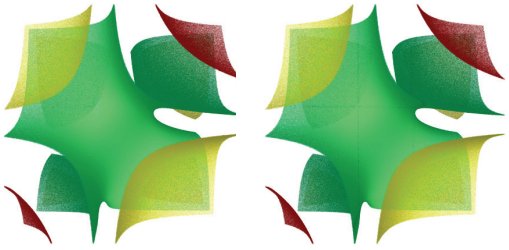
ζ=0.1, sampling time 35 [sec]



ζ=0.01, sampling time 102 [sec]



ζ=0.01, sampling time 56 [sec]



ζ=0.001, sampling time 191 [sec]

図 6.3　ζの値に対する提案手法の適用結果（左列）。近似関数により生成された粒子を黒色にしたものが右列。

ζ=0.001, sampling time 84 [sec]

図 6.4　ζの値に対する提案手法の適用結果（左列）。近似関数により生成された粒子を黒色にしたものが右列。

しかしながら、データのセル数が多くなり、画面に対するセルの大きさが相対的に小さくなるような場合には、図 6.4 に見られるようなアーティファクトは非常に目立ちにくくなると考えられる。このことについて検証するため、解像度 120x120x34 の構造格子データ“Lobster”に対して、提案手法、従来の PBVR 向けサンプリング手法(一点積分とメトロポリスサンプリング)、レイキャスティング(等間隔サンプリング)、等値面を適用し、画質の比較を行う。ζの値を 1 とする。伝達関数は不透明度に非常に鋭いピークを一つだけ与え、そのスカラー値の範囲にほぼ一致するように等値面のしきい値を与えた。結果を図 6.5 に示す。この結果から、提案手法が最も等値面に近い画像を生成することが確認された。そして、近似関数に起因するアーティファクトは確認されなかった。

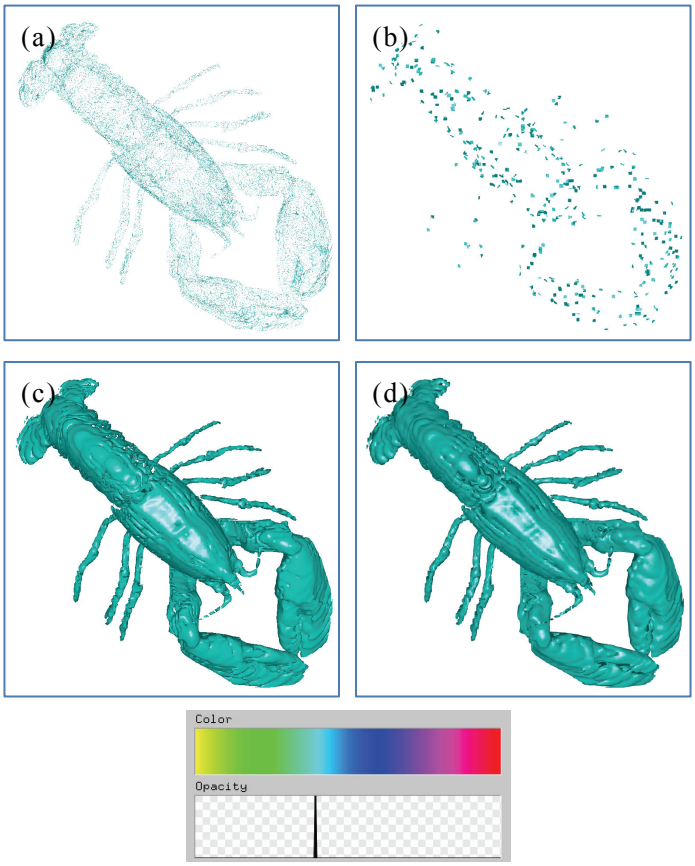


図 6.5　Lobster データに対する適用結果。(a)レイキャスティング法、(b)従来の PBVR 向けサンプリング手法、(c)等値面、(d)提案手法、最下段は適用した伝達関数。

6.5.3 適用例

六面体向けレイヤードサンプリングを実データに適用し、画質の検証とサンプリング時間の計測を行う。パラメータの設定として、ガウスールジャンドル積分は、分割数 2、分点数 10 を採用する。逆関数法の近似のためのパラメータはζ=1とする。

画質と計算速度の検証をするため、六面体向けレイヤードサンプリングと従来の PBVR（一点積分とメトロポリスサンプリング）、そしてレイキャスティング法を構造格子からなるメディカルデータ“Kidney”（解像度 200x200x141）に適用する。図 6.6 はレンダリング結果と適用した伝達関数を示す。従来の PBVR とレイキャスティングによる結果画像からはアーティファクトが、そして提案手法はそれを改善していることが確認できる。

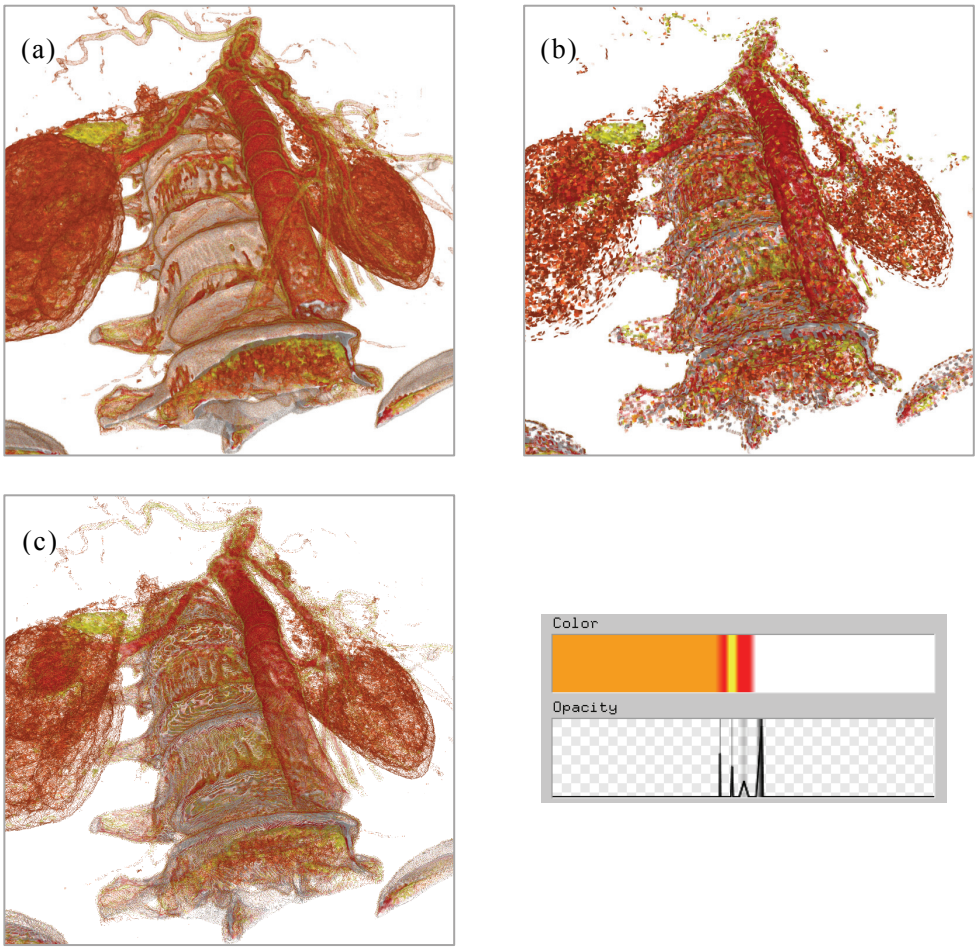


図 6.6　４つのピークを持つ伝達関数に対するメディカルデータのレンダリング結果。(a)提案手法、(b)従来の PBVR 向けサンプリング手法、(c)レイキャスティング、右下は適用した伝達関数。提案手法は白く色付された骨と赤い血管、黄色く色付された膜をレンダリングすることに成功している。他の手法では、アーティファクトが見られる。

従来手法と提案手法を不規則六面体格子からなるデータ“Oral”に適用し、画質の検証とサンプリング時間の計測を行う。Oral は口腔流体シミュレーションの結果であり、頂点数 2,725,232、セル数 2,646,268 である。図 6.7 にレンダリング結果を示す。この結果から、伝達関数が急峻に変化する青色と黄色の領域に対して、従来手法はレンダリングアーティファクトが見られ、提案手法はそれを改善している事がわかる。また、伝達関数の変化があまり激しくない領域に関しては、従来手法でも十分な画質が得られていることが確認できる。このことから、提案手法は六面体からなるボリュームデータに対して、従来手法よりも高い画質の粒子生成を行えることが言える。

表 6.4 に Kidney と Oral に対するサンプリング時間と積分時間を示す。この結果から、提案手法は従来手法よりも多くのサンプリング時間を要することが解る。また、Oral データに関して、提案手法のサンプリング時間のほとんどが積分計算の時間で占められていた。提案手法により粒子生成を行ったのち、従来の PBVR と同様にレンダリングを行っている。レンダリングスピードに関して、提案手法は従来手法よりも多くの粒子を生成しているため、粒子数に反比例してレンダリングスピードが下がっているが、著しい低下なく高画質にレンダリングすることに成功している。

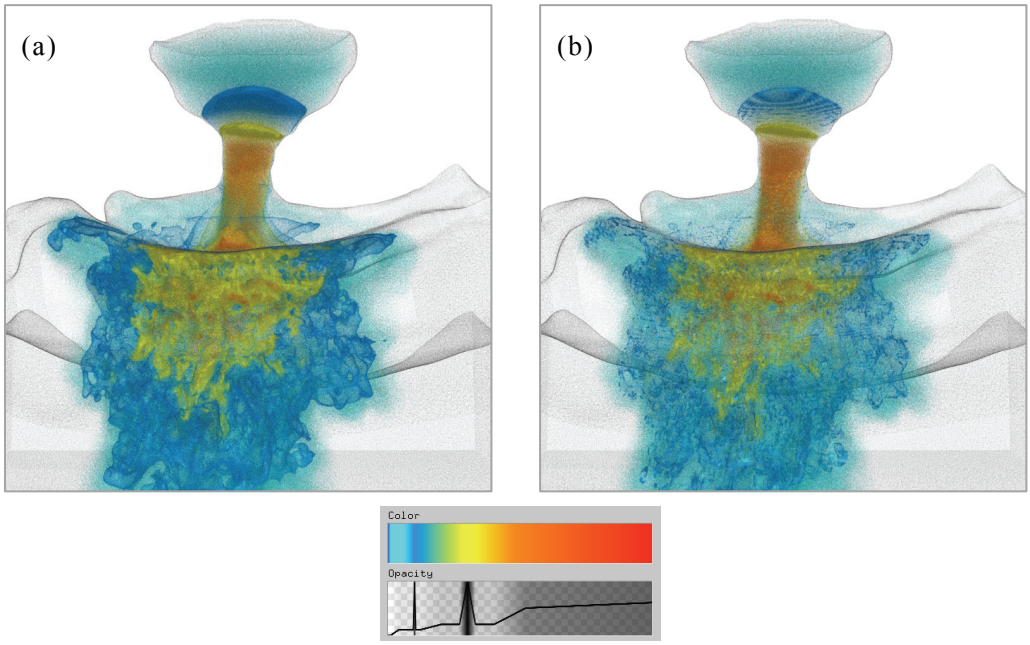


図 6.7 口腔流体シミュレーションデータの可視化結果。(a)提案手法、(b)従来の PBVR 向けサンプリング手法、下段が使用した伝達関数。灰色の曲面はデータの境界面であり、境界面上で粒子生成を行い可視化している。変化が鋭い青と黄色の領域について、従来手法ではアーティファクトが見られる。

表 6.4　Kidney 及び Oral データに適用した結果の積分時間とサンプリング時間。

	Kidney		Oral	
セル数	5,544,140		2,646,268	
サンプリング手法	提案手法	従来手法	提案手法	従来手法
積分時間	264 [sec]		426	
サンプリング時間	877 [sec]	40 [sec]	61.3 [sec]	4.7 [sec]
生成粒子数	45.8 [M]	27.3 [M]	4.8 [M]	1.5 [M]
FPS	3.0	3.7	8.7	11.2

6.6　まとめ

従来の PBVR は、不規則六面体格子に対して急峻に変化する伝達関数が与えられた場合に、高品質な粒子生成を行うことができなかった。そこで四面体格子向けレイヤードサンプリングで用いられる、区分線形伝達関数に対して、制御点の作る各区分で粒子生成を行い、高品位な粒子生成を行うという枠組みを取り入れ、それを不規則六面体格子向けに拡張した、六面体格子向けレイヤードサンプリングを提案した。この手法は生成粒子数を求めるための体積積分の手法として、積分領域の格子分割を併用したガウスールジャンドル積分を用いた。各区分で粒子生成を行うため、スカラー値そのものを媒介変数として含む変数変換を行った。変数変換されたスカラー場に対して効率よく粒子生成を行うため、ギブス法を用いた。そしてギブス法において計算するべき 1 変数の密度関数に対して、累積積分関数の逆関数を陽に計算することができたため、逆関数法を適用した。提案手法は六面体を要素に持つ、規則格子と不規則六面体格子の両方に対して画質の改善が見られた。しかしながら、メトロポリスサンプリングと一点積分を用いた従来手法に比べて、サンプリング時間が長いという問題点も見られた。

六提案手法の数値積分の精度とパフォーマンスの検証を行うため、格子の分割数とガウスールジャンドル積分の分点数を変化させて計算を行い、計算速度の計

測を行った。そして積分値とパフォーマンスに関する比較を行うために、モンテカルロ積分による計算も行った。その結果、計算精度は分点数と分割数の積に関係し、その値が大きいほど精度が高いという知見が得られた。分割数と分点数の積が等しい場合は、サンプリング数も等しくなるが、サンプリング点の位置が異なることに留意する。そして、分割ガウスールジャンドル法はモンテカルロ積分と同程度の精度の計算を行う場合、モンテカルロ積分よりも高速であることが確認できた。

逆関数法で行っている関数の近似が画質と粒子生成速度に与える影響を検証するため、近似の度合いを表すパラメータを変化させて画質と計算時間を比較した。その結果、近似の度合いが小さくなるほど計算時間が増大し、画質は向上した。また、ボリュームデータの格子数が多くなると、相対的に画面に対する格子が小さくなり、近似関数によるアーティファクトは見られないと言う事がわかった。

提案手法を実データに適用して、画質の検証とサンプリング時間の計測を行った。従来手法の PBVR で生成された画像にはアーティファクトが確認されたのに対し、提案手法はそれを改善できた。しかし提案手法の粒子生成時間は従来の PBVR に比べて大きかった。

提案手法では、生成粒子数の計算に関して、形状関数のヤコビ行列式$|\det \mathbf{J}_s|$を評価している。粒子群生成の際にも同様にし$|\det \mathbf{J}_s|$を評価するべきであるが、提案手法ではそれを行わず、局所座標内部でのみ粒子密度分布が正しくなるように計算を行っている。このことは、六面体の異方性が高いときに、画質に影響を及ぼすと予想される。将来的には、この問題を解決した手法を開発する予定である。

提案手法は粒子生成をセル毎に独立して行うことができるため、並列化による高速化が容易であると考えられる。また、より高速かつ高精度な積分手法を考案し、積分時間の短縮も行いたい。また、2 次以上の要素を持つ複雑な格子データへの適用できる、高精度なサンプリング手法の開発が望まれる。

第7章　結論

本研究では、画質に関する問題点を解決し、大規模な不規則格子ボリュームデータに適用するための粒子生成手法や、メモリ効率のよい画像生成手法を提案した。

伝達関数に対して画質が一意に定まらないという問題点を解決するために、密度発光モデルから粒子密度推定式を導出し、生成粒子数を計算する手法を提案した。この手法を用いて、ボリュームレンダリングでは高画質な手法として知られるレイキャスティング法と比較可能な画像を生成することができた。サブピクセルレベルを増加させるほど、レイキャスティングの画像に近づくことがわかった。また、この手法は確率統計的な手法で粒子を生成しているため、画像にゆらぎが生じる。このゆらぎについて検討するため、結果画像の画素値の分散を計算した。その結果、サブピクセルレベルの増加に伴う分散の減少を確認できた。そして、画像の分散の値を計算する仮定で作成された平均の画像について、画質の改善が確認された。この手法は従来手法に置き換わるものではなく、従来手法ではレンダリングが困難であったメモリに乗り切らないサイズのデータを、プレビュー用にボリュームレンダリングすることができる手法である。この手法を四面体格子からなるボリュームデータに適用し、10 億格子からなる四面体格子のボリュームレンダリングに成功した。

従来の PBVR は四面体以外の不規則格子に対して粒子のサンプリングを行うことができなかった。この問題を解決するために、粒子密度推定式を用いて PBVR を不規則格子に適用するための粒子分布生成手法を提案した。この手法は有限要素法で用いられるアイソパラメトリック要素に対する補間関数と形状関数を利用して、局所座標内部でサンプリングを行い、大域座標へマッピングすることで粒子生成を行った。そして、格子内部での粒子密度分布を生成するために、局所座標内部でメトロポリスサンプリングを行った。

サブピクセル処理はサブピクセルレベルを増やせば高い画質が得られる反面、フレームバッファのサイズが大きくなりメモリを圧迫すると言う問題点があった。画像生成時の PBVR のメモリ効率を改善するため、画像重畳を用いてレンダ

リング画像を生成する、リピート処理を提案した。リピート処理はサブピクセル処理の代わりに、確率的に得られた画像の平均を計算することで画質が向上するという、PBVR の画像のゆらぎの解析から得られた結果を用いていた。更に、リピート処理を応用して画像詳細度制御を実現した。スムーズな対話的操作を目的としたこの技術は、データを動かす時には粗く高速にレンダリングを、止めると詳細にレンダリングを行った。画質について検証するためリピート処理とサブピクセル処理の画像を比較した結果､同等の画質の画像が得られることがわかった。また使用したフレームバッファの大きさを計測した結果、リピート処理によりメモリ使用量を抑えられることが確認できた。提案手法を有限要素法で計算された実データに適用し、従来手法ではボリュームレンダリングが困難だった大規模不規則格子ボリュームデータ（領域分割された 7200 万の六面体要素や、2600 万の四面体二次要素から構成されたボリュームデータ）をボリュームレンダリングすることに成功した。また、粒子生成に一様分布を用いた以前の PBVR と画質を比較した結果、メトロポリスサンプリングの効果により、ブロッキーノイズが軽減されていることが確認できた。

ユーザーがある特定のスカラー値の範囲に着目してボリュームレンダリングを行う場合、伝達関数はそのスカラー値に対応した不透明度の範囲だけが大きくなるように調整される。この範囲が狭まるに従って伝達関数は急峻になる。このとき、発生粒子数の計算と粒子生成を正確に行うことが困難であり、画像にアーティファクトを生じた。PBVR による画像は、粒子密度推定法により計算されるボリュームデータ内部の粒子密度分布に依存するため、適切な生成粒子数の計算が必要となる。しかし従来の PBVR は生成粒子数の計算に、格子の密度に関する一点積分を用いていた｡そのため急峻な伝達関数に対してサンプリングが失敗し、積分精度が下がっていた。この問題を解決するために、四面体格子に適用可能な高品位な粒子生成手法である四面体向けのレイヤードサンプリング法を提案した。レイヤードサンプリングでは、区分線形伝達関数から区分線形密度関数を作成し、それを参照してレギュラーボックスと呼ばれる計算空間内部に粒子を事前生成する｡体積積分と粒子生成は事前生成粒子群を用いて行われる｡実験により、粒子数を計算するための体積積分の精度と画質､粒子生成時間について検証した。積分の精度について、単純な形状と線形に分布するスカラー場について厳密解を計算し、モンテカルロ積分（従来手法はモンテカルロ積分の特別な場合）と比較した。その結果、モンテカルロよりも高精度な結果が得られた。また、実データ

に適用し画質と粒子生成時間を比較した。従来手法に顕著に見られた等値面領域に関するアーティファクトを、レイヤードサンプリングで改善することができた。しかしレイヤードサンプリングの粒子生成は従来手法よりも遅く、ボリュームデータの格子数が多いほどその差が大きくなった。

四面体向けのレイヤードサンプリングを拡張し、六面体向けレイヤードサンプリングを提案した。この手法は、マルコフ連鎖モンテカルロ法の一種であるギブスサンプリングと、逆関数法の組み合わせから構成されており、区分線形伝達関数を構成する各区分（レイヤー）において粒子生成を行った。逆関数法に用いる関数が特異点を持っていたため、特異点回りに近似関数を与えて計算を行った。この近似関数は近似の度合いを現すパラメータを持ち、計算精度を制御することができた。また、従来の PBVR では生成粒子数を計算するために、格子中心における一点積分を行っていたが、本手法ではより高精度に計算を行うために、ガウスールジャンドル積分を適用した。提案手法のガウスールジャンドル積分は、積分点の増加と格子の分割と併用した。提案手法で粒子生成を行った後に、従来の PBVR の枠組みでレンダリングを行った。六提案手法の数値積分の精度とパフォーマンスの検証を行うため、格子の分割数とガウスールジャンドル積分の分点数を変化させて計算を行い、計算速度の計測を行った。そして積分値とパフォーマンスに関する比較を行うために、モンテカルロ積分による計算も行った。その結果、計算精度は分点数と分割数の積に関係し、その値が大きいほど精度が高いという知見が得られた。分割数と分点数の積が等しい場合は、サンプリング数も等しくなるが、サンプリング点の位置が異なった。そして、分割ガウスールジャンドル法はモンテカルロ積分と同程度の精度の計算を行う場合、モンテカルロ積分よりも高速であることが確認できた。逆関数法で行っている関数の近似が画質と粒子生成速度に与える影響を検証するため、近似の度合いを現すパラメータを変化させて画質と計算時間を比較した。その結果、近似の度合いが小さくなるほど計算時間が増大し、画質は向上した。また、ボリュームデータの格子数が多くなると、相対的に画面に対する格子が小さくなり、近似関数によるアーティファクトは見られないと言う事がわかった。提案手法を実データに適用して、画質の検証とサンプリング時間の計測を行った。従来手法の PBVR で生成された画像にはアーティファクトが確認されたのに対し、提案手法はそれを改善できた。しかし提案手法の粒子生成時間は従来の PBVR に比べて大きかった。

今後の課題として、粒子生成時間の短縮が必要とされる。PBVR は前処理とし

て粒子の生成を行い、ユーザーが伝達関数を変更するたびに粒子の再生成が行われる。粒子生成時間は生成粒子数とデータの格子数に比例して増加し、特にレイヤードサンプリングでは粒子生成時間が大きい。PBVR は格子毎に粒子生成を行うため、GPU を用いた並列処理による高速化が期待される。

PBVR はレンダリングプリミティブを粒子化することによって、様々な統合表示が可能となる。例えば、口腔流体データの可視化では、データの境界面上に粒子を生成して、ボリュームレンダリングと境界面との統合表示を行っている。この技術を応用して、スカラー場とベクトル場、テンソル場の統合表示手法の開発を計画している。また、時系列データ向け PBVR 手法は未だに提案されておらず、その効率的な手法の開発も今後の課題である。

計算機性能や数値シミュレーション技術の発達に伴いボリュームデータの大規模・複雑化が進んでおり、それらに対する可視化技術の開発は非常に重要である。しかし、格子の探索やソート処理がメモリ容量に関する、または計算速度に関するボトルネックとなり、規則格子の可視化と比べて明らかに研究が遅れていた。本研究は大規模不規則格子ボリュームデータを効率よく扱うための原理やフレームワークを与えるものであり、今後の大規模不規則格子に対する可視化手法を開発する上での基盤となることが期待される。

謝辞

本研究を進めるにあたり、豊富なアイデアや活発な議論により、有益な御指導、御助言を頂きました京都大学高等教育研究開発推進センター小山田耕二教授に深く感謝申し上げます。京都大学高等教育研究開発推進センター特定助教授の坂本尚久氏には、多忙な中、論文の執筆や、プログラミング技術などを教えて頂き、本研究にも多大な御指導、御協力を頂きました。大阪大学の野崎先生には、口腔流体シミュレーションデータの提供のみならず、様々な流体物理学に関するアドバイスをいただき、流体データの可視化手法に関しての有益な議論ができました。東京大学の奥田先生から頂いた構造解析データは、本研究において非常に重要な役割を果たしました。

日ごろより学生生活を共にしてきた小山田研究室学生の皆様からは、時に新たな観点からの議論や助言を頂くことがありました。深く感謝いたします。

本研究について貴重な意見や、ご指摘を頂きました、生体機能工学分野の小林哲夫教授と学術情報メディアセンター中村祐一教授に深く感謝致します。

引用文献

[1] B. H. McCormick, T. A. Defanti, and M. D. Brown, "Visualization in Scientific Computing," 1987.

[2] P. Sabella, "A Rendering Algorithm for Visualizing 3D Scalar Fields," *Computer Graphics*, vol. 22, no. 4, pp. 51-58, 1988.

[3] Naohisa Sakamoto, Jorge Nonaka, Koji Koyamada, and Satoshi Tanaka, "Particle-based Volume Rendering," in *Asia-Pacific Symposium on Visualization*, Sydney, Australia, 2007, pp. 141–144.

[4] Koji Koyamada, Naohisa Sakamoto, and Satoshi Tanaka, "A Particle Modeling for Rendering Irregular Volumes," in *International Conference on Computer Modeling and Simulation*, Emmanuel College, Cambridge, England, 2008, pp. 372–377.

[5] Donald Shepard, "A two-dimensional interpolation function for irregularly-spaced data," in *Proceedings of the 1968 ACM National Conference*, 1968, pp. 517–524.

[6] Steven Martin, Han-Wei Shen, and Ravi Samtaney, "Efficient Rendering of Extrudable Curvilinear Volumes," in *Visualization Symposium, 2008. PacificVIS '08. IEEE Pacific*, Kyoto, Japan, 2008, pp. 1 - 8.

[7] Koji Koyamada and T. Nishio, "Volume visualization of 3D finite element method results," *IBM Journal of Research and Development*, vol. 35, no. 12, pp. 12-25, 1991.

[8] Takuma Kawamura, Naohisa Sakamoto, and Koji Koyamada, "A Streamline Visualization Technique for Sub-volume Based CFD Results," in *Asia Simulation Conference*, 2009, p. (Poster).

[9] W.E. Lorensen and H.E. Cline, "Marching cubes: A high resolution 3D surfaceconstruction algorithm," *Computer Graphics*, vol. 21, no. 4, pp. 163–169, 1987.

[10] Jim Blinn, "Light Reflection Function for Simulation of Clouds and Dusty Surfaces," *Computer Graphics*, vol. 16, no. 3, pp. 21-29, 1982.

[11] James T. Kajiya and Brian P Von Herzen, "Ray Tracing Volume Densities," *Computer Graphics*, vol. 18, no. 3, pp. 165-174, 1984.

[12] Marc Levoy, "Display of Surfaces from Volume Data," *IEEE Computer Graphics and Applications*, vol. 8, no. 3, pp. 29-37, 1988.

[13] D. Shreiner, M. Woo, J. Neider, and T. Davis, *OpenGL Programming Guide (5th Edition)*: Addison-Wesley, 2005.

[14] Peter Shirley and Allan Tuchman, "A Polygonal Approximation to Direct Scalar Volume Rendering," in *San Diego Workshop on Volume Visualization*, San Diego, 1990, pp. 63-70.

[15] Klaus Engel, Martin Kraus, and Thomas Ertl, "High-Quality Pre-Integrated

Volume Rendering Using Hardware-Accelerated Pixel Shading," in *Eurographics/SIGGRAPH Workshop on Graphics Hardware '01*, 2001, pp. 9-16.

[16] Steven P. Callahan, Milan Ikits, Joao L. D. Comba, and Claudio T. Silva, "Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, no. 3, pp. 285-295, 2005.

[17] Koji Koyamada, "Volume Visualization for the Unstructured Grid Data," in *SPIE/SPSE Symposium on Electronic Imaging, Conference Proceedings No. 1259*, 1990, pp. 14-25.

[18] M. P. Garrity, "Raytracing irregular volume data," *Computer Graphics*, vol. 24, no. 5, pp. 35-40, 1990.

[19] Koji Koyamada, Sakae Uno, Akio Doi, and Tatsuo Miyazawa, "Fast Volume Rendering by Polygonal Approximation," *Journal of Information Processing*, vol. 15, no. 4, pp. 535-544, 1991.

[20] P. Williams, "Visibility-ordering of Meshed Polyhedra," *Transaction on Graphics*, vol. 11, no. 2, pp. 103-126, 1992.

[21] B. Lucas et al., "An Architecture for a Scientific Visualization System," in *IEEE Visualization*, 1992, pp. 107-113.

[22] Koji Koyamada and Takayuki Itoh, "Fast generation of spherical slicing surfaces for irregular volume rendering," *Visual Computer*, vol. 11, no. 3, pp. 167-175, 1995.

[23] J. Meredith and Kwan-Liu Ma, "Multiresolution View-Dependent Splat-based Volume Rendering of Large Irregular Data," in *IEEE 2001 Symposium On Parallel and Large-Data Visualization and Graphics*, 2001, pp. 93-155.

[24] B. Wylie, K. Moreland, L. A. Fisk, and P. Crossno, "Tetrahedral Projection using Vertex Shaders," in *IEEE Symposium on Volume Visualization* , 2002, pp. 7-12.

[25] S. Roettger and T. Ertl, "Cell Projection of Convex Polyhedra," in *Volume Graphics* , 2003, pp. 103-107.

[26] Balázs Csébfalvi and László Szirmay-Kalos, "Monte Carlo Volume Rendering," in *IEEE Visualization* , 2003, pp. 449-456.

[27] Balázs Csébfalvi, "Interactive Transfer Function Control for Monte Carlo Volume Rendering," in *IEEE Symposium on Volume Visualization and Graphics* , 2004, pp. 33-38.

[28] Erik W. Anderson, Steven P. Callahan, Carlos E. Scheidegger, John M. Schreiner, and Claudio T. Silva, "Hardware-Assisted Point-Based Volume Rendering of Tetrahedral Meshes," in *Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI)*, Washington, DC, USA, 2007, pp. 163-172.

[29] Philipp Muigg, Markus Hadwiger, Helmut Doleisch, and Helwig Hauser, "Scalable Hybrid Unstructured and Structured Grid Raycasting," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1592-1599, 2007.

[30] Naohisa Sakamoto and Koji Koyamada, "Particle-based volume rendering," *可視化情報学会論文集*, vol. 27, no. 2, pp. 7–14, 2007.

[31] Joachim Georgii and Rüdiger Westermann, "A Generic and Scalable Pipeline for GPU Tetrahedral Grid Rendering," *IEEE TRANSACTIONS ON VISUALIZATION*

AND COMPUTER GRAPHICS, vol. 12, no. 5, pp. 1345-1352, 2006.

[32] Fábio F. Bernardon, Steven P. Callahan, João L. D. Comba, and Cláudio T. Silva, "An Adaptive Framework for Visualizing Unstructured Grids with Time-Varying Scalar Fields," *Parallel Computing*, vol. 33, no. 6, pp. 391-405, 2007.

[33] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross, "EWA Volume Splatting," in *IEEE Visualization*, 2001, pp. 29-36.

[34] Wei Chen, Liu Ren, Matthias Zwicker, and Hanspeter Pfister, "Hardware-Accelerated Adaptive EWA Volume Splatting," in *IEEE Visualization*, 2004, pp. 67-74.

[35] Yuan Zhou and Michael Garland, "Interactive Point-Based Rendering of Higher-Order Tetrahedral Data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1229–1236, 2006.

[36] Stéphane Marchesin and Guillaume Colin de Verdi`ere, "High-Quality, Semi-Analytical Volume Rendering for AMR Data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1611-1618, 2009.

[37] Aaron Knoll et al., "Volume Ray Casting with Peak Finding and Differential Sampling," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1571-1578, 2009.

[38] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller, "Equationof state calculations by fast computing machines," *Journal of ChemicalPhysics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[39] S. Mase, J. Møller, D. Stoyan, R. Waagepetersen, and G. Döge, "Packing Densities and Simulated Tempering for Hard Core Gibbs Point Processes," *Annals of the Institute of Statistical Mathematics*, vol. 53, no. 4, pp. 661-680, 2001.

[40] M. Matsumoto and T. Nishimura, "Mersenne twister: 623-dimensionally quidistributed," *ACM Transaction on Modelingand Computer Simulation*, vol. 8, no. 1, pp. 3-30, 1998.

[41] George Marsaglia, "Xorshift RNGs," *Journal of Statistical Software*, vol. 8, no. 14, pp. 1-4, 2003.

[42] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen, "Decimation of Triangle Meshes," in *SIG-GRAPH* , Chicago, 1992, pp. 65-70.

[43] H. Hoppe, "Progressive Meshes," in *SIG-GRAPH* , New Orleans, 1996, pp. 99-108.

[44] 中村浩子, 竹島由里子, 藤代一成, and 奥田洋司, "形状/色分布特徴を考慮した区間型ボリュームの半自動詳細度制御," *情報処理学会論文誌*, vol. 42, no. 5, pp. 1115-1123, 2001.

[45] Steven Molnar, Michael Cox, David Ellsworth, and Henry Fuchs, "A sorting classification of," *IEEE Computer Graphics and Applications*, vol. 14, no. 4, pp. 23–32, 1994.

[46] 文部科学省次世代 IT 基盤構築のための研究開発「革新的シミュレーションソフトウェアの研究開発」プロジェクト. [Online]. http://www.rss21.iis.u-tokyo.ac.jp/

[47] 河村拓馬, 坂本尚久, 山崎晃, and 小山田耕二, "粒子ベースボリュームレンダリングのための粒子密度推定法 -大規模非構造ボリュームデータに対する適用-," *可視化情報学会論文誌*, vol. 27, no. 2, pp. 69-77, 2008.

[48] O. G. Staadt and M. H. Gross, "Progressive Tetrahedralizations," in *IEEE Visualiza-tion* , 1998, pp. 99-108.

[49] Takuma Kawamura, Naohisa Sakamoto, and Koji Koyamada, "Level-of-detail rendering of a large-scale irregular volume dataset using particles," *JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY*, vol. 25, no. 5, pp. 905-915, Sept 2010.

[50] Takuma Kawamura, Koji Koyamada, Naohisa Sakamoto, and Satoshi Tanaka, "A High-Quality Sampling Technique of PBVR for Unstructured Hexahedral Mesh Data," *IEEE Visualization'10 (Poster)*, 2010.

[51] Takuma Kawamura, Naohisa Sakamoto, and Koji Koyamada, "A High Quality Sampling Technique for Particle-based Volume Rendering," *IEEE Visualization'09 (Poster)*, 2009.